





□ Prefácio

As presentes folhas foram elaboradas com o intuito de dar apoio à exposição da matéria lecionada na unidade curricular de Computação Física, da Licenciatura em Engenharia de Informática e Multimédia (LEIM), do ISEL.

Estas folhas devem ser utilizadas em conjunto com as que foram elaboradas pelo Prof. Jorge Pais, servindo de complemento às mesmas, no sentido de fornecer ao aluno mais informação sobre os temas abordados na unidade curricular de Computação Física.

Carlos Carvalho, 2022



□ Componentes de avaliação

- Componente prática – três (3) trabalhos práticos
 - A nota da componente prática corresponde à média aritmética das notas dos trabalhos práticos. Não existe uma nota mínima para cada trabalho, mas a componente prática tem uma nota mínima, que é de 9,5 valores.
- Componente teórica – um (1) exame escrito (duas épocas + época especial)
 - A nota da componente teórica corresponde à nota obtida no exame, com a nota mínima de 9,5 valores.

A nota final da unidade curricular é a média aritmética das duas componentes.

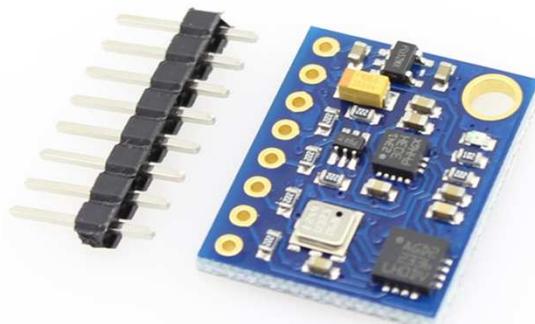


☐ Material a adquirir

Para o 3.º trabalho prático, será necessária a aquisição do seguinte material:

- Placa com magnetómetro, acelerómetro, giroscópio e sensor de pressão atmosférica, com interface I²C
- *Display* alfanumérico de 16 colunas por 2 linhas (16×2), com interface I²C

Chipset: L3GD20, LSM303D e BMP180
(10DOF for Arduino, ou GY-89)





ECTS

Créditos ECTS da unidade curricular de CF: **6 créditos**

Horas de trabalho para alcançar os objetivos do programa de estudos:

- $6 \text{ créditos} \times 26,6(6) \text{ horas/crédito} = \textbf{160 horas}$
- Componente letiva:
 - $4,5 \text{ horas / semana} \times 15 \text{ semanas} = \textbf{67,5 horas}$
 - Componente de estudo para o exame:
 - $25 \text{ horas} \times 1 \text{ Exame} = \textbf{25 horas}$
 - Componente de estudo e trabalho fora de aulas:
 - $(160 - 67,5 - 25) / 15 \text{ semanas} = \textbf{4,5 horas/semana}$



Números e bases numéricas



□ Conceito de número

- Um número é um conjunto de símbolos (algarismos), com um determinado **peso** cada, destinados a designar uma quantidade ou um código.
- A numeração romana não é ponderada, na medida em que os seus símbolos não têm uma significância posicional.

Números naturais (conjunto \mathbb{N})

Para já, irão considerar-se os números inteiros não negativos. Este conjunto recebe o nome de números naturais, porque é o conjunto que se aprende em primeiro lugar, e o mais simples.



□ Sistemas de numeração

- Código numérico com significância posicional
- Em cada posição, ter-se-á um algarismo, cujo peso tem a ver com essa posição e cuja quantificação é relativa à base numérica desse número.
- A base determina a variedade de algarismos (símbolos) necessários.
- Os sistemas de numeração têm regras operatórias sobre as quantidades representadas.
- As operações realizam-se sobre os algarismos do mesmo peso, provocando arrasto para o peso seguinte.



□ Significância posicional

$$N = \sum_{i=0}^{L-1} A_i B^i = A_0 B^0 + A_1 B^1 + \cdots + A_{L-1} B^{L-1}$$

N – Valor a representar

B – Base de numeração

L – Comprimento (número de algarismos de N)

A_i – Algarismos do número, contendo apenas símbolos possíveis em B

B^i – Peso, ou significância, do algarismo A_i

i – Índice de iteração, indo da direita para a esquerda, começando em 0

Exemplo: $(1987)_{10} = 7 \times 10^0 + 8 \times 10^1 + 9 \times 10^2 + 1 \times 10^3 = 7 + 80 + 900 + 1000 = 1987$



□ Bases de numeração mais utilizadas

- Base 10 (*decimal*) – Base que utilizamos no dia a dia, contendo dez algarismos (símbolos), de 0 a 9.
- Base 2 (*binário*) – Base utilizada em cálculos digitais. Têm-se dois símbolos: 0 e 1. Cada dígito nesta base chama-se *bit* – *binary digit*. Na verdade, o bit é uma medida de informação.
- Base 8 (*octal*) – “Auxiliar” da base 2. Oito símbolos: 0 a 7.
- Base 16 (*hexadecimal*) – “Auxiliar” da base 2. Dezasseis símbolos: 0 a F.

Em qualquer base, de acordo com o comprimento do número utilizado, tem-se uma capacidade máxima de representação que é B^L .



□ Conversão entre bases de numeração

Tendo um número, este é representável de forma diferente consoante a base.

A conversão para decimal faz-se diretamente pela significância posicional.

- Conversão binário → decimal:

$$(0110)_2 \rightarrow N = 0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 = 2 + 4 = (6)_{10}$$

- Conversão octal → decimal :

$$(614)_8 \rightarrow N = 4 \times 8^0 + 1 \times 8^1 + 6 \times 8^2 = 4 + 8 + 6 \times 64 = (396)_{10}$$

- Conversão hexadecimal → decimal:

$$(6A4)_{16} \rightarrow N = 4 \times 16^0 + 10 \times 16^1 + 6 \times 16^2 = 4 + 160 + 6 \times 256 = (1700)_{10}$$



□ Conversão entre bases de numeração

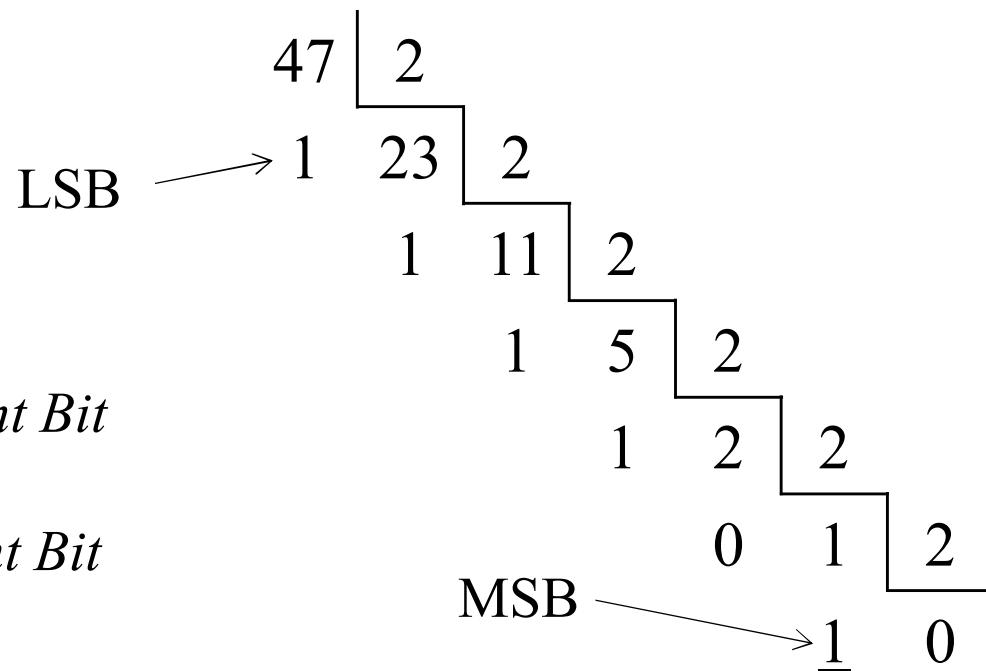
A conversão de base 10 para a base 2 faz-se através de divisões sucessivas por 2, até obter quociente 0. Vão-se aproveitando os restos das divisões.

- Conversão decimal → binário :

$$(47)_{10} \rightarrow (101111)_2$$

MSB – *Most Significant Bit*

LSB – *Least Significant Bit*





□ Conversão entre bases de numeração

A conversão de base 10 para as bases 8 e 16 faz-se através de divisões sucessivas por 8 ou 16, respetivamente, até obter quociente 0. Vão-se aproveitando os restos das divisões.

- Conversão decimal → octal :

$$(47)_{10} \rightarrow (57)_8$$

$$\begin{array}{r} 47 \quad | \quad 8 \\ \underline{7} \quad \quad | \quad 5 \quad | \quad 8 \\ \underline{\underline{5}} \quad \quad \quad 0 \end{array}$$

A – 10

B – 11

C – 12

D – 13

E – 14

F – 15

- Conversão decimal → hexadecimal :

$$(47)_{10} \rightarrow (2F)_{16}$$

$$\begin{array}{r} 47 \quad | \quad 16 \\ \underline{15} \quad \quad | \quad 2 \quad | \quad 16 \\ \underline{\underline{2}} \quad \quad \quad 0 \end{array}$$



□ Conversão entre bases de numeração

A conversão de base 2 para as bases 8 e 16 faz-se, agrupando o número em conjuntos de 3 ou 4 bits, respetivamente. Dentro de cada um, converte-se como se fosse para decimal.

- Conversão binário → octal :

$$(10011011)_2 \rightarrow (\underbrace{010}_{} \underbrace{011}_{} \underbrace{011}_{})_2 = (233)_8$$

- Conversão binário → hexadecimal :

$$(1001011011)_2 \rightarrow (\underbrace{0010}_{} \underbrace{0101}_{} \underbrace{1011}_{})_2 = (25B)_{16}$$



□ Conversão entre bases de numeração

A conversão de base 8 ou base 16, para base 2, realiza-se fazendo corresponder um algarismo a um grupo de 3 ou 4 bits, respetivamente.

- Conversão octal → binário :

$$(4765)_8 \rightarrow (\underbrace{100}_4 \underbrace{111}_7 \underbrace{110}_6 \underbrace{101}_5)_2$$

- Conversão hexadecimal → binário :

$$(2EB)_{16} \rightarrow (\underbrace{0010}_2 \underbrace{1110}_{E(14)} \underbrace{1011}_{B(11)})_2$$



☐ Arduino e bases de numeração

Na linguagem de programação do Arduino existe a possibilidade de se especificarem números nas bases numéricas já abordadas.

Por *default*, a base numérica de trabalho é a decimal.

Base	Formato	Exemplo	Comentário
10 (decimal)	nenhum	123	
2 (binário)	Prefixo “B”	B110111	Até 8 bits de comprimento. Só 0 e 1 são válidos.
8 (octal)	Prefixo “0”	0765	Só caracteres 0-7 são válidos.
16 (hexadecimal)	Prefixo “0x”	0xF2E8	Só caracteres 0-9 e A-F (ou a-f) são válidos.



☐ Arduino e bases de numeração

No *serial monitor*, para especificar a escrita de números de acordo com uma determinada base numérica, devem utilizar-se os seguintes modificadores na chamada ao método `Serial.print()` e `Serial.println()`.

Base	Exemplo	Resultado
10 (decimal)	<code>Serial.print(78, DEC)</code>	78
2 (binário)	<code>Serial.print(78, BIN)</code>	1001110
8 (octal)	<code>Serial.print(78, OCT)</code>	116
16 (hexadecimal)	<code>Serial.print(78, HEX)</code>	4E



□ Operação soma entre dois números binários

Esta operação é idêntica à praticada em base 10, havendo arrasto (*carry*) para o peso seguinte, sempre que se ultrapasse a capacidade do peso corrente.

Exemplo: Somar $A = 100101$ (37_{10}) com $B = 110111$ (55_{10}) $\rightarrow S = (92)_{10}$

$$\begin{array}{ccccccccc} A & = & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 & = & 1 & 0 & 0 & 1 & 0 & 1 \\ B & = & B_5 & B_4 & B_3 & B_2 & B_1 & B_0 & = & 1 & 1 & 0 & 1 & 1 & 1 \end{array}$$
$$\begin{array}{ccccccccc} & (C_4) & (C_3) & (C_2) & (C_1) & (C_0) & & & \\ & 0 & \leftarrow & 0 & \leftarrow & 1 & \leftarrow & 1 & \leftarrow \\ A & & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ + B & & + & 1 & 1 & 0 & 1 & 1 & + \\ \hline S & & 1 & 0 & 1 & 1 & 0 & 1 & 92 \end{array}$$

The diagram shows the binary addition of $A = 100101$ and $B = 110111$ to produce $S = 1001010$. The result is shown below the summands with carries labeled above each column. Carries are labeled $(C_4), (C_3), (C_2), (C_1), (C_0)$ from left to right. Arrows point from each carry label to its corresponding column. The carries are 0, 0, 1, 1, 1 respectively. The final sum is 92.

Olhando a cada índice (peso), de uma forma geral, tem-se $S_i = A_i + B_i + C_{i-1}$



□ Números relativos (conjunto \mathbb{Z})

- Numa subtração, no caso do subtrativo ser superior ao aditivo, não existe representação sob a forma de número natural. Por este motivo, surgiu o conjunto dos números relativos \mathbb{Z} , englobando positivos e negativos.
- Noção de números simétricos: são dois números, tais que somados um com o outro, o resultado é igual a zero.
- Nos computadores e, em geral, nas máquinas de cálculo, existe uma capacidade máxima de representação numérica porque o *hardware* é finito. Essa representação ($n.^o$ de dígitos – L), deverá contemplar tanto os números positivos como os negativos.



□ Código de complementos (complemento para dois)

Obtenção de números simétricos em binário

Algoritmo 1: Complementar todos os bits (complemento para 1) e somar 1.

Algoritmo 2: Percorrer o número da direita para a esquerda até encontrar o primeiro 1, mantê-lo, e inverter todos os bits que se seguem até chegar à extremidade esquerda.

Exemplos: 0010 → 1110 Representação a 4 bits

 (+2) → (-2) Pesos em complemento para 2:

-8	4	2	1
----	---	---	---

 0101 → 1011 Pesos em binário natural:
 (+5) → (-5) Pesos em binário natural:

8	4	2	1
---	---	---	---



☐ Código de complementos (complemento para dois)

Em binário, ter-se-á uma representação finita a L bits.

$$L = 4 : 2^4 = 16 \text{ configurações}$$

- 8 positivas: 0 a +7: 0000 a 0111

- 8 negativas: -1 a -8: 1111 a 1000

O dígito de maior peso indica o sinal algébrico do número:

0, se for positivo
1, se for negativo

→ bit de sinal

+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000



□ Representação com um número finito de bits

Representação com $4 / L$ bits:

- Binário natural (só “positivos”):

$$\{0, \dots, 2^4 - 1\} = \{0, \dots, 15\}$$

$$\{0, \dots, 2^L - 1\}$$

- Código de complementos (positivos e negativos):

$$\{-2^{(4-1)}, \dots, 0, \dots, +2^{(4-1)} - 1\} = \{-8, \dots, 0, \dots, +7\}$$

$$\{-2^{(L-1)}, \dots, 0, \dots, +2^{(L-1)} - 1\}$$

		\mathbb{N}		\mathbb{Z}
15	1111		+7	0111
14	1110		+6	0110
13	1101		+5	0101
12	1100		+4	0100
11	1011		+3	0011
10	1010		+2	0010
9	1001		+1	0001
8	1000		0	0000
7	0111		-1	1111
6	0110		-2	1110
5	0101		-3	1101
4	0100		-4	1100
3	0011		-5	1011
2	0010		-6	1010
1	0001		-7	1001
0	0000		-8	1000



☐ Representação com um número finito de bits

- Extensão de sinal:

Em representações de maiores dimensões, os números positivos ficam com os bits à esquerda todos a zero e, os negativos, com esses bits todos a 1.

Exemplos:

Número	Representação a 4 bits	Representação a 16 bits (int)
+3	0011	0000000000000011
+1	0001	0000000000000001
-8	1000	1111111111111100
-3	1101	1111111111111101
-1	1111	1111111111111111



□ Subtração entre números binários

- Subtração direta
- Adição ao simétrico do subtrativo (é assim que o *hardware* opera)

Exemplos com subtração direta e adição com simétrico do subtrativo:

$$\begin{array}{r} 7 \\ - 3 \\ \hline + 4 \end{array} \quad \begin{array}{r} 0 & 1 & 1 & 1 \\ - 0 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 0 \end{array}$$

$$\begin{array}{r} 0 & 1 & 1 & 1 & (+7) \\ + 1 & 1 & 0 & 1 & (-3) \\ \hline 1 & 0 & 1 & 0 & (+4) \end{array}$$

$$\begin{array}{r} 2 \\ - 5 \\ \hline - 3 \end{array} \quad \begin{array}{r} 0 & 0 & 1 & 0 \\ - 0 & 1 & 0 & 1 \\ \hline 1 & 1 & 0 & 1 \end{array}$$

$$\begin{array}{r} 0 & 0 & 1 & 0 & (+2) \\ + 1 & 0 & 1 & 1 & (-5) \\ \hline 0 & 1 & 1 & 0 & (-3) \end{array}$$

CBN: $Bw = 1$. Deu 13 e ficou a dever 16.
CBC: O resultado deu -3. Correto.

Carry = 1: é desprezado, pois fica fora da representação.

Borrow = 0: não fica a dever nada. Arrasto, numa subtração, para um peso de ordem superior. Resultado correto.

CBN: $Cy = 0 \rightarrow Bw = 1$ (O resultado deu 13 - incoerente)
CBC: O resultado deu -3. Correto.



☐ Indicadores de erro em código binário natural

Como a capacidade de representação do *hardware* é finita, poderão haver resultados decorrentes de somas ou de subtrações que não consigam ficar dentro dessa representação.

Indicadores de erro:

Código binário natural

Operação soma: **Carry (Cy)**

Operação subtração: **Borrow (Bw)**

Estes ocorrerão sempre que o resultado não consiga ficar dentro da representação, originando um resultado incoerente com os operandos.



□ Indicador de erro em código binário de complementos

Em código de complementos, caso o resultado decorrente de um cálculo não se situe dentro da gama de valores permitidos, não existe representação possível. Tal significa que o resultado apresentado nesse número finito de bits será incoerente com a operação e o valor dos operandos.

Indicador de erro (operações soma e subtração): *Overflow (Ov)*

Por definição, o *overflow* ocorre se, tendo como operandos dois números positivos, o resultado for negativo, e vice-versa.

Sempre que se somar (ou subtrair) números de sinais algébricos opostos, o resultado ficará sempre dentro da gama de representação possível.



□ Indicadores de erro em código binário natural e de complementos

Consoante se esteja a operar no domínio dos números naturais ou no dos números relativos, o número (código binário) resultante de uma operação terá o respetivo significado nesse domínio. Caso o resultado de uma operação aritmética “pretenda” dar um número fora da gama possível, tal será prontamente sinalizado pelo indicador de erro desse domínio.

Habitualmente, o *hardware* realiza as subtrações à custa da adição com o simétrico do subtrativo.

Vejamos alguns exemplos de situações e respetivo resultado e indicação de erro (se for o caso).



□ Exemplos de somas

Soma

$(\mathbb{N})_{10}$

$(\mathbb{Z})_{10}$

$$\begin{array}{r} A = \quad \begin{array}{rrrr} 1 & 0 & 0 & 0 \end{array} \\ + B = \quad \begin{array}{rrrr} + & 0 & 1 & 1 \end{array} \\ \hline 0 \quad \underbrace{1 \quad 1 \quad 1 \quad 1} \end{array} \quad \begin{array}{r} 8 \\ + 7 \\ \hline 15 \end{array} \quad \begin{array}{r} (-8) \\ + (+7) \\ \hline (-1) \end{array} \quad \begin{array}{l} Cy = 0 \\ Ov = 0 \end{array}$$

$$\begin{array}{r} A = \quad \begin{array}{rrrr} 1 & 0 & 1 & 0 \end{array} \\ + B = \quad \begin{array}{rrrr} + & 1 & 0 & 1 \end{array} \\ \hline 1 \quad \underbrace{0 \quad 1 \quad 0 \quad 1} \end{array} \quad \begin{array}{r} 10 \\ + 11 \\ \hline 5 \end{array} \quad \begin{array}{r} (-6) \\ + (-5) \\ \hline (+5) \end{array} \quad \begin{array}{l} Cy = 1 \\ Ov = 1 \end{array}$$



□ Exemplos de somas

Soma

$(\mathbb{N})_{10}$

$(\mathbb{Z})_{10}$

$$\begin{array}{r} A = \quad 0 \ 1 \ 1 \ 0 \\ + B = + 0 \ 1 \ 1 \ 0 \\ \hline 0 \ \underline{1} \ 1 \ 0 \ 0 \end{array} \qquad \begin{array}{r} 6 \\ + 6 \\ \hline 12 \end{array} \qquad \begin{array}{r} (+6) \\ + (+6) \\ \hline (-4) \end{array} \qquad \begin{array}{l} Cy = 0 \\ Ov = 1 \end{array}$$

$$\begin{array}{r} A = \quad 1 \ 0 \ 1 \ 0 \\ + B = + 1 \ 1 \ 1 \ 0 \\ \hline 1 \ \underline{1} \ 0 \ 0 \ 0 \end{array} \qquad \begin{array}{r} 10 \\ + 14 \\ \hline 8 \end{array} \qquad \begin{array}{r} (-6) \\ + (-2) \\ \hline (-8) \end{array} \qquad \begin{array}{l} Cy = 1 \\ Ov = 0 \end{array}$$



□ Exemplos de subtrações

Subtração

$$\begin{array}{r} A = \quad 0 \ 1 \ 1 \ 0 \\ - B = \quad - 0 \ 0 \ 0 \ 1 \\ \hline 0 \ 0 \ 1 \ 0 \ 1 \\ \downarrow \end{array}$$

$$\begin{array}{r} A = \quad 0 \ 1 \ 1 \ 0 \\ \overline{B} = \quad 1 \ 1 \ 1 \ 0 \\ + \quad 0 \ 0 \ 0 \ 1 \\ \hline 1 \ 0 \ 1 \ 0 \ 1 \end{array}$$

$(\mathbb{N})_{10}$

$$\begin{array}{r} 6 \\ - 1 \\ \hline 5 \end{array}$$

$(\mathbb{Z})_{10}$

$$\begin{array}{r} (+6) \\ - (+1) \\ \hline (+5) \end{array}$$

Quando a subtração é feita por intermédio de uma soma, o Cy daí resultante deve ser transformado em Bw .

$$Cy = 1 \rightarrow Bw = 0$$



□ Exemplos de subtrações

Subtração

$$\begin{array}{r} A = \quad 0 \ 1 \ 1 \ 0 \\ - B = - 0 \ 1 \ 1 \ 1 \\ \hline 1 \ 1 \ 1 \ 1 \\ \downarrow \end{array}$$

$$\begin{array}{r} A = \quad 0 \ 1 \ 1 \ 0 \\ \overline{B} = \quad 1 \ 0 \ 0 \ 0 \\ + \quad 0 \ 0 \ 0 \ 1 \\ \hline 0 \ 1 \ 1 \ 1 \end{array}$$

$(\mathbb{N})_{10}$

$(\mathbb{Z})_{10}$

$$\begin{array}{r} 6 \\ - 7 \\ \hline 15 \end{array}$$

$$\begin{array}{r} (+6) \\ - (+7) \\ \hline (-1) \end{array}$$

Na subtração direta, o bit mais à esquerda, que fica fora da representação, é diretamente o indicador (*flag*) de *Borrow*.

$$Cy = 0 \rightarrow \boxed{Bw = 1}$$



□ Exemplos de subtrações

Subtração

$(\mathbb{N})_{10}$

$(\mathbb{Z})_{10}$

$$\begin{array}{r} A = \quad 1 \ 0 \ 1 \ 0 \\ - B = - 0 \ 1 \ 1 \ 1 \\ \hline 0 \ 0 \ 1 \ 1 \end{array} \qquad \begin{array}{r} 10 \\ - 7 \\ \hline 3 \end{array} \qquad \begin{array}{r} (-6) \\ - (+7) \\ \hline (+3) \end{array} \qquad \begin{array}{l} Bw = 0 \\ Ov = 1 \end{array}$$

$$\begin{array}{r} A = \quad 1 \ 0 \ 1 \ 0 \\ \overline{B} = \quad 1 \ 0 \ 0 \ 0 \\ + \quad 0 \ 0 \ 0 \ 1 \\ \hline 1 \ 0 \ 0 \ 1 \ 1 \end{array}$$

$$Cy = 1 \rightarrow Bw = 0$$



□ Exemplos de subtrações

Subtração

 $(\mathbb{N})_{10}$ $(\mathbb{Z})_{10}$

$$\begin{array}{r} A = \quad 0 \ 1 \ 1 \ 1 \\ - B = - 1 \ 0 \ 1 \ 0 \\ \hline \end{array} \qquad \begin{array}{r} 7 \\ - 10 \\ \hline 13 \end{array} \qquad \begin{array}{r} (+7) \\ - (-6) \\ \hline (-3) \end{array} \qquad \begin{array}{l} Bw = 1 \\ Ov = 1 \end{array}$$

A diagram shows arrows pointing from the circled '1' in the first subtraction to the circled '0' in the second subtraction, and from the circled '1' in the second subtraction to a box containing the equation $Cy = 0 \rightarrow Bw = 1$.

$$\begin{array}{r} A = \quad 0 \ 1 \ 1 \ 1 \\ \overline{B} = \quad 0 \ 1 \ 0 \ 1 \\ + \quad 0 \ 0 \ 0 \ 1 \\ \hline \end{array}$$

$$Cy = 0 \rightarrow Bw = 1$$



Circuitos combinatórios



□ Operações lógicas

- Existem três operações lógicas elementares – NOT, AND e OR.
- A cada operação está atribuído um operador, tal que se podem escrever expressões lógicas à base destas operações elementares.
- A partir das operações lógicas elementares, é possível definir outras operações (NAND, NOR, XOR e XNOR), que embora não sendo elementares, podem ser tratadas como tal, com o devido conhecimento das regras a que obedecem.



□ Operação lógica NOT (negação lógica)

Definição: Operação sobre *uma* variável (ou termo, ou função), de que resulta a inversão do seu valor lógico.

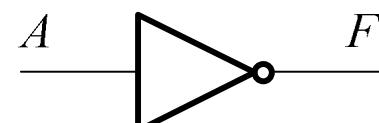
Tabela de verdade

A	F
0	1
1	0

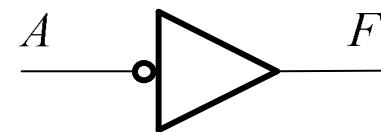
Expressão algébrica

$$F = \bar{A}$$

Símbolo lógico



ou





□ Operação lógica NOT (negação lógica)

Propriedades:

$$\overline{\overline{A}} = A \quad \text{Teorema da involução}$$
$$\overline{\overline{\overline{A}}} = \overline{A}$$



□ Operação lógica AND (produto lógico)

Definição: Operação sobre n variáveis, que só toma o valor 1 quando todas essas variáveis tiverem o valor 1.

Tabela de verdade

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Expressão algébrica

$$F = A \cdot B$$

Símbolo lógico





□ Operação lógica AND (produto lógico)

Propriedades:

$A \cdot 0 = 0$ Elemento absorvente

$A \cdot 1 = A$ Elemento neutro

$A \cdot A = A$ Teorema da idempotência

$A \cdot \bar{A} = 0$ Teorema da complementação

$A \cdot B = B \cdot A$ Propriedade comutativa

$(A \cdot B) \cdot C = A \cdot (B \cdot C) = A \cdot B \cdot C$ Propriedade associativa



□ Operação lógica OR (soma lógica)

Definição: Operação sobre n variáveis, que só toma o valor 0 quando todas essas variáveis tiverem o valor 0.

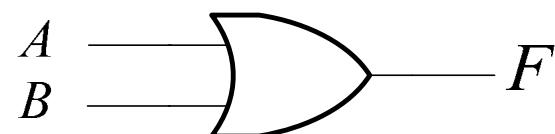
Tabela de verdade

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Expressão algébrica

$$F = A + B$$

Símbolo lógico





□ Operação lógica OR (soma lógica)

Propriedades:	$A + 0 = A$	Elemento neutro
	$A + 1 = 1$	Elemento absorvente
	$A + A = A$	Teorema da idempotência
	$A + \bar{A} = 1$	Teorema da complementação
	$A + B = B + A$	Propriedade comutativa
	$(A + B) + C = A + (B + C) = A + B + C$	Propriedade associativa



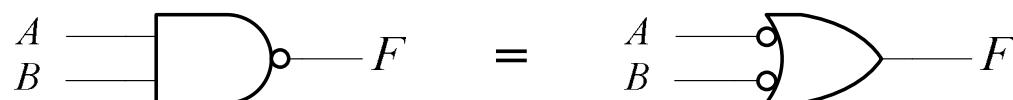
□ Teoremas de De Morgan

Propriedades:

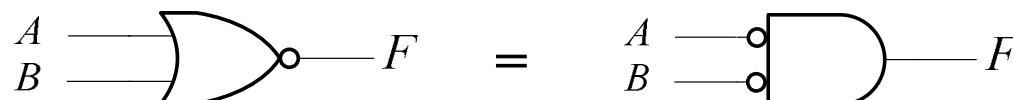
$$\begin{array}{l} \overline{A \cdot B \cdot C} = \overline{A} + \overline{B} + \overline{C} \\ \hline \overline{A + B + C} = \overline{A} \cdot \overline{B} \cdot \overline{C} \end{array}$$

Exemplos de aplicação dos teoremas de De Morgan:

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$



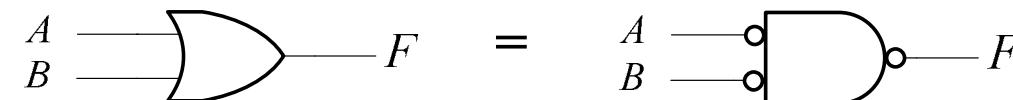
$$\overline{A + B} = \overline{A} \cdot \overline{B}$$



$$A \cdot B = \overline{\overline{A} \cdot \overline{B}} = \overline{\overline{A} + \overline{B}}$$



$$A + B = \overline{\overline{A} + \overline{B}} = \overline{\overline{A} \cdot \overline{B}}$$





□ Forma AND-OR

Considere-se, a título de exemplo, a função lógica dada pela sua tabela de verdade. Extraiendo a função pelos 1's, sem simplificação, fica:

n	C	B	A	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

$$F(C,B,A) = \bar{C}\bar{B}A + \bar{C}B\bar{A} + \bar{C}BA + C\bar{B}\bar{A} + C\bar{B}A + CBA$$

Forma canónica AND-OR (união de intersecções – termos produto):

- Em cada termo constam todas as variáveis da função, complementadas ou não complementadas;
- Os termos produto em que intervêm todas as variáveis da função denominam-se “termos mínimos”.

Representação alternativa (apenas válida com a forma canónica):

$$F(C,B,A) = \sum(1,2,3,4,5,7)$$



□ Forma OR-AND

Considere-se, a título de exemplo, a mesma função lógica dada anteriormente.

Extraindo a função pelos 0's, sem simplificação, fica:

n	C	B	A	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

$$F(C, B, A) = \overline{\overline{C} \cdot \overline{B} \cdot \overline{A}} + C \cdot B \cdot \overline{A} = (C + B + A) \cdot (\overline{C} + \overline{B} + A)$$

Forma canónica OR-AND (interseção de uniões – termos soma):

- Em cada termo constam todas as variáveis da função, complementadas ou não complementadas;
- Os termos soma em que intervêm todas as variáveis da função denominam-se “termos máximos”.

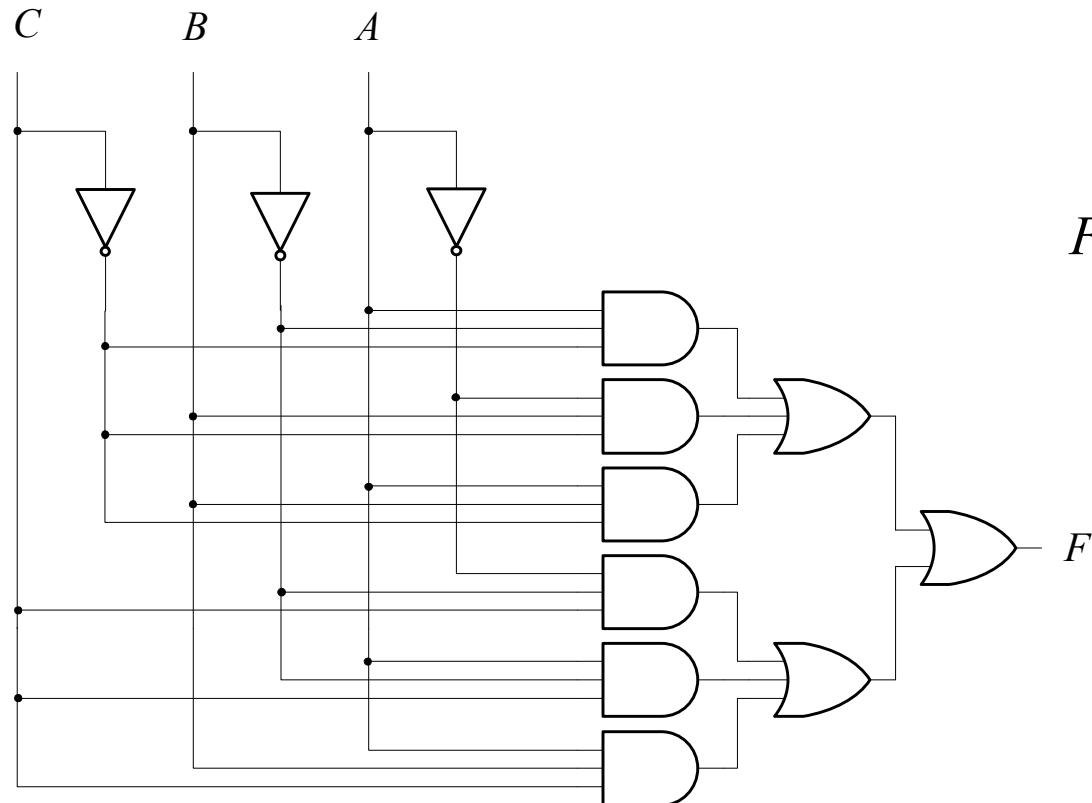
Representação alternativa (apenas válida com a forma canónica):

$$F(C, B, A) = \prod (0,6)$$

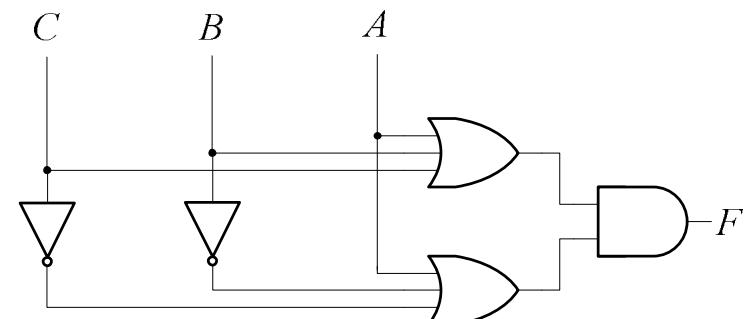


☐ Implementação do circuito lógico sob as formas AND-OR e OR-AND

$$F(C, B, A) = \overline{C} \cdot \overline{B} \cdot A + \overline{C} \cdot B \cdot \overline{A} + \overline{C} \cdot B \cdot A + C \cdot \overline{B} \cdot \overline{A} + C \cdot \overline{B} \cdot A + C \cdot B \cdot A$$



$$F(C, B, A) = (C + B + A) \cdot (\overline{C} + \overline{B} + \overline{A})$$





□ Propriedades das formas AND-OR

Em simplificação algébrica, os critérios de prioridade das várias operações são:

- O símbolo AND tem prioridade sobre o OR e o XOR;
- As operações entre parêntesis têm prioridade face às que estão fora deles.

$$A + B \cdot C = (A + B) \cdot (A + C) \quad - \text{Propriedade distributiva}$$

Exemplos de aplicação de propriedades:

$$A + A \cdot B = A \cdot (1 + B) = A$$

$$A \cdot B + A \cdot C = A \cdot (B + C)$$

$$A + \overline{A} \cdot B = A \cdot (1 + B) + \overline{A} \cdot B = A + A \cdot B + \overline{A} \cdot B = A + B \cdot (A + \overline{A}) = A + B$$



□ Propriedades das formas OR-AND

Em simplificação algébrica, os critérios de prioridade das várias operações são:

- Idem, como na forma AND-OR.

$$A.(B + C) = A.B + A.C \quad - \text{Propriedade distributiva}$$

Exemplos de aplicação de propriedades:

$$A.(A + B) = A + A.B = A$$

$$A.(\bar{A} + B) = A.B$$

$$(A + B).(A + C) = A + A.C + A.B + B.C = A + B.C$$

$$(A + B).(C + D) = A.C + A.D + B.C + B.D$$



□ Mapas de Karnaugh

Dada a morosidade da simplificação algébrica, torna-se necessário utilizar outro método de simplificação de funções lógicas.

n	C	B	A	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Funções não simplificadas, na sua forma canónica:

AND-OR

$$F(C, B, A) = \overline{C} \cdot \overline{B} \cdot A + \overline{C} \cdot B \cdot \overline{A} + \overline{C} \cdot B \cdot A + C \cdot \overline{B} \cdot \overline{A} + C \cdot \overline{B} \cdot A + C \cdot B \cdot A$$

OR-AND

$$F(C, B, A) = (C + B + A) \cdot (\overline{C} + \overline{B} + A)$$



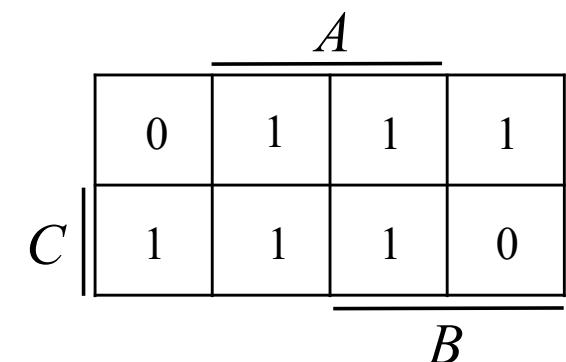
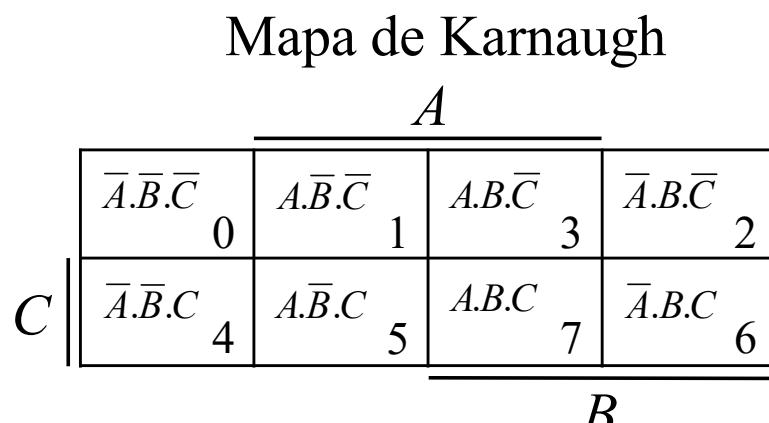
□ Mapas de Karnaugh

$$F(A, B, C) = A \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot B \cdot \bar{C} + A \cdot B \cdot \bar{C} + \bar{A} \cdot \bar{B} \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot C$$

$$F(A, B, C) = (A + B + C) \cdot (A + \bar{B} + \bar{C})$$

n	C	B	A	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Representação alternativa da tabela de verdade





□ Características dos mapas de Karnaugh

- Tanto o diagrama de Venn como o mapa de Karnaugh são formas alternativas de representar a tabela de verdade, logo têm 2^L possibilidades (áreas) distintas.
- De cada uma das áreas do mapa de Karnaugh, para uma qualquer adjacente (ou simétrica), só muda o valor de uma única variável.
- Dentro da quadrícula do mapa coloca-se o valor lógico de saída da função para a configuração de variáveis de entrada respeitante a essa quadrícula.

		A			
		$\bar{A}.\bar{B}.\bar{C}$	$A.\bar{B}.\bar{C}$	$A.B.\bar{C}$	$\bar{A}.B.\bar{C}$
		0	1	3	2
C	$\bar{A}.\bar{B}.C$	4	5	7	6
	$A.\bar{B}.C$				



□ Procedimento de simplificação utilizando mapas de Karnaugh

- A simplificação consiste em agrupar os “1”s, adjacentes ou simétricos, em grupos cuja quantidade seja potência inteira de dois. Por cada agrupamento de 2^x “1”s, reduzem-se x variáveis no respetivo termo produto.
- Deve tentar agrupar-se o máximo número de “1”s.
- “1”s já presentes num agrupamento podem também constar de outro, se daí se obtiver uma maximização no número de “1”s em cada grupo.

		A			
		$\bar{A}.\bar{B}.\bar{C}$	$A.\bar{B}.\bar{C}$	$A.B.\bar{C}$	$\bar{A}.B.\bar{C}$
		0	1	3	2
C	$\bar{A}.\bar{B}.C$	4	5	7	6
	$A.\bar{B}.C$				

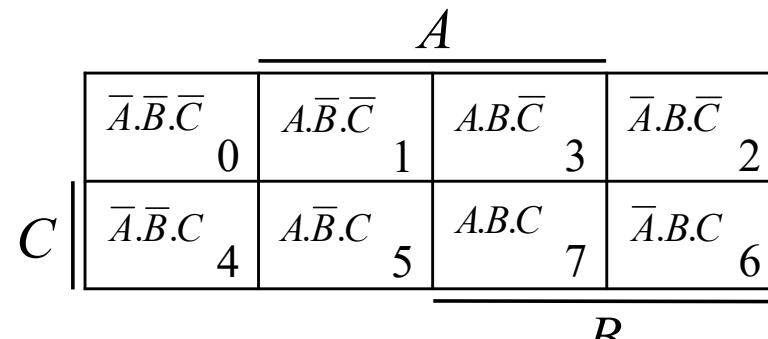


□ Procedimento de simplificação utilizando mapas de Karnaugh

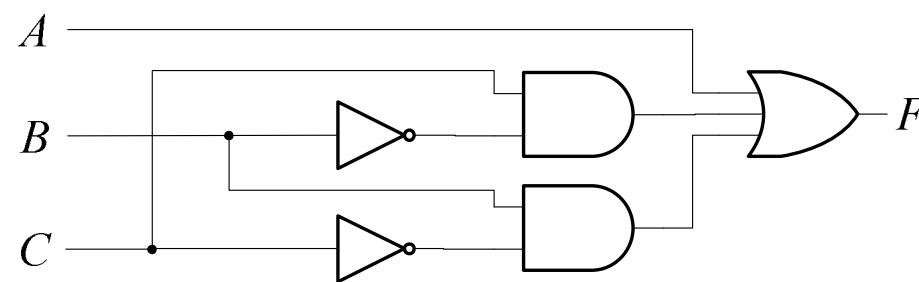
$$F(A, B, C) = A \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot B \cdot \bar{C} + A \cdot B \cdot \bar{C} + \bar{A} \cdot \bar{B} \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot C$$

$$F(A, B, C) = (A + B + C) \cdot (A + \bar{B} + \bar{C})$$

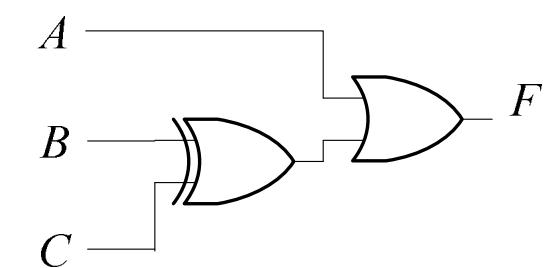
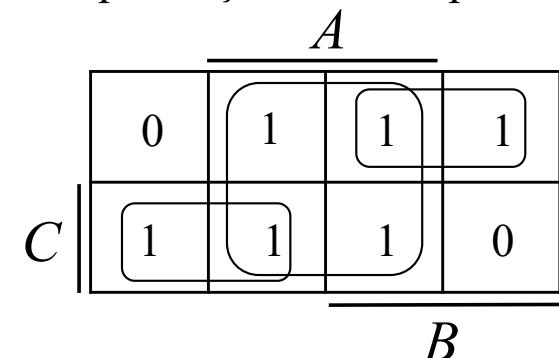
n	C	B	A	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1



$$F = A + \bar{B} \cdot C + B \cdot \bar{C} = A + (B \oplus C)$$



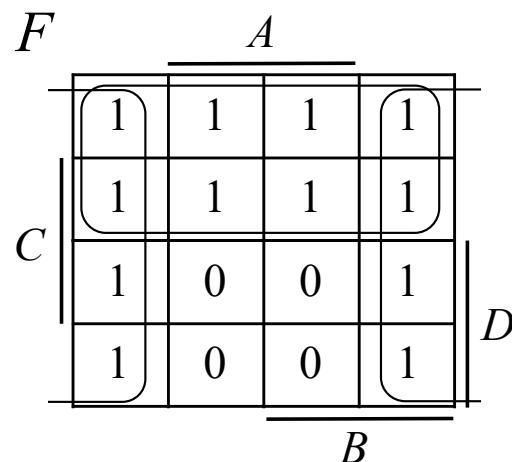
Simplificação do exemplo dado:



□ Exemplo de simplificação

$$F = \overline{A}.\overline{C} + \overline{A}.D + \overline{A}.B + A.\overline{D} + B.\overline{D}$$

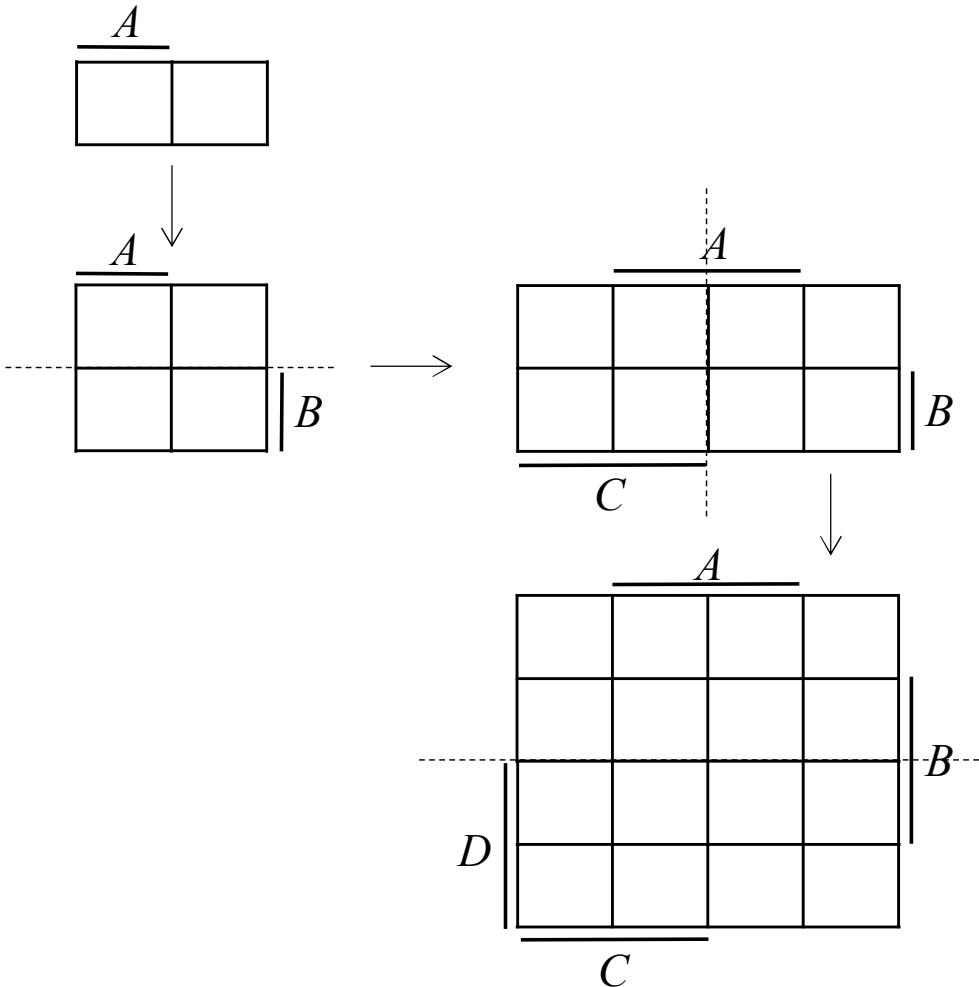
Quatro variáveis: 16 quadrículas



A disposição das variáveis no mapa é indiferente. Contudo, tem de ser sempre respeitada a regra da variação de apenas uma variável entre quadrículas adjacentes e entre quadrículas simétricas.



□ Obtenção dos mapas



1 variável: 2 quadrículas

2 variáveis: 4 quadrículas

3 variáveis: 8 quadrículas

4 variáveis: 16 quadrículas

Geralmente, para mais de 4 variáveis, utilizam-se processos auxiliares para simplificação com mapas (e.g. variáveis inseridas)

Caso geral: n variáveis $\rightarrow 2^n$ quadrículas



□ Exemplos

A	B	C	D	X	Y
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	0

O conselho diretivo de uma escola é formado por 4 membros que têm de votar sobre assuntos de gestão da mesma. Pretende-se um circuito combinatório mínimo que acenda um led verde (X) quando a maioria dos votantes votar afirmativamente e um led vermelho (Y) quando a maioria votar negativamente uma determinada decisão.

X C

A	<hr/>			
	0	0	0	0
	0	0	1	0
	0	1	1	1
	1	0	0	1
	1	0	0	0
	1	0	1	0
	1	0	1	1
	1	1	0	0
	1	1	0	1
	1	1	1	0
	1	1	1	1

D

Y C

A	<hr/>			
	1	1	0	1
	1	0	0	0
	0	0	0	0
	1	0	0	0

D

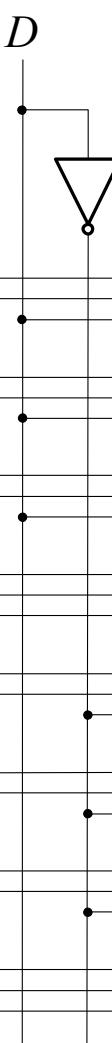
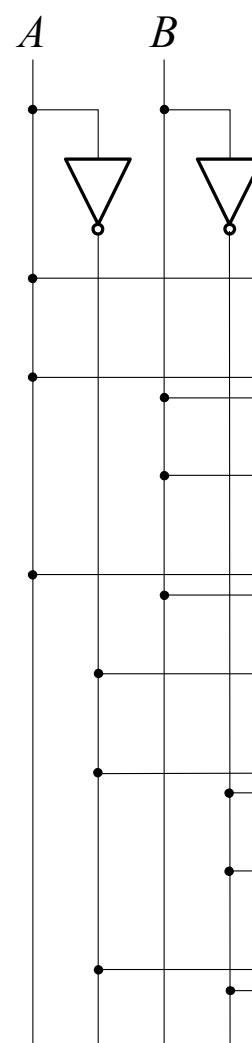
$$X = A.C.D + A.B.D + B.C.D + A.B.C$$

$$Y = \overline{A}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{D} + \overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}$$



□ Exemplos

A	B	C	D	X	Y
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	0



$$X = A.C.D + A.B.D + B.C.D + A.B.C$$

$$Y = \bar{A}.\bar{C}.\bar{D} + \bar{A}.\bar{B}.\bar{D} + \bar{B}.\bar{C}.\bar{D} + \bar{A}.\bar{B}.\bar{C}$$



□ Exercícios

Construir a tabela de verdade e o mapa de Karnaugh de cada uma das seguintes funções lógicas e obter a expressão simplificada para cada função dada:

$$F_1 = A \cdot B + \overline{A} \cdot B + \overline{A} \cdot \overline{B}$$

$$F_2 = \overline{\overline{A} + \overline{B} + \overline{C}}$$

$$F_3 = \overline{B} + A \cdot B \cdot \overline{C}$$

$$F_4 = \overline{A \cdot B} + A \cdot B \cdot \overline{C}$$

$$F_5 = A \cdot \overline{C} + B \cdot \overline{C} + A \cdot C + \overline{B} \cdot C$$

$$F_6 = x \cdot y \cdot z + x \cdot y \cdot \overline{z} + x \cdot \overline{y} \cdot z + x \cdot \overline{y} \cdot \overline{z}$$

$$F_7 = A \cdot \overline{D} + C \cdot B \cdot A \cdot \overline{D} + \overline{C} \cdot \overline{B} + \overline{C}$$

$$F_8 = \overline{x \cdot w \cdot y \cdot z + \overline{z} + \overline{x}}$$



□ Funções incompletamente especificadas

Existem funções em que é indiferente o valor lógico de saída, em função de certas combinações das variáveis de entrada.

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	×
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	×

B	<u>A</u>			
	1	1	1	1
0	0	0	×	×

C

× - indiferente, “*don't care*”. Pode tomar qualquer valor (1 ou 0), dependendo de como resultar uma maior simplificação.

Considerando $\times = 1$: $F = \overline{B} + C$

Considerando $\times = 0$: $F = \overline{B}$

Neste caso, tornava-se mais vantajoso tomar “ \times ” como igual a 0.

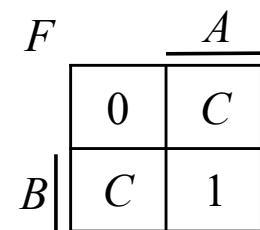


☐ Tabelas de verdade e mapas de Karnaugh com variáveis inseridas

No caso de se terem mais do que quatro variáveis de entrada, pode-se adotar uma estratégia que torna o processo de simplificação muito mais simples do que se se utilizassem tabelas maiores e, consequentemente, mapas maiores.

- Suponha-se uma situação em que dois indivíduos, A e B , têm que estar de acordo sobre um dado assunto, sendo F a decisão tomada. Se estiveram ambos de acordo que “não” (0) a decisão será 0. Se estiverem ambos de acordo que “sim” (1), a decisão será 1. Caso não haja consenso entre os dois, um terceiro elemento, C , é chamado a decidir.

A	B	F
0	0	0
0	1	C
1	0	C
1	1	1





□ Mapas de Karnaugh com variáveis inseridas

A extração de expressões simplificadas em mapas de Karnaugh com variáveis inseridas faz-se em duas etapas:

- 1) Extrair os “1”s presentes no mapa, considerando como “0” as variáveis inseridas, e aproveitando os eventuais *don't care* para simplificação;
- 2) Extrair as variáveis inseridas, considerando também os “1”s como *don't care*.

Para o exemplo anterior, fica:

F	\overline{A}
0	C
B	
C	1

$$F = \underbrace{A \cdot B}_{1)} + \underbrace{A \cdot C}_{2)} + B \cdot C$$

Na etapa 2), pelo facto dos agrupamentos serem feitos com “ C ”s e não com “1”s, a variável binária C irá fazer parte do termo de cada agrupamento.



□ Mapas de Karnaugh com variáveis inseridas

Para tornar mais claro, veja-se a extração das expressões simplificadas do mapas de Karnaugh, em cada uma das etapas:

1) F'

	A
B	0 0
	0 1

$$F' = A \cdot B$$

2) F''

	A
B	0 C
	C -

$$F'' = A \cdot C + B \cdot C$$

O *don't care* transforma-se em C , ficando-se com grupos de “ C ”s.

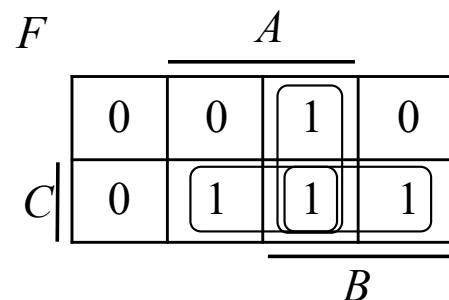
$$F = F' + F'' = \underbrace{A \cdot B}_{1)} + \underbrace{A \cdot C + B \cdot C}_{2)}$$



□ Mapas de Karnaugh com variáveis inseridas

Para demonstrar a validade da técnica apresentada, o mesmo cenário do exemplo anterior também pode ser descrito por uma tabela e um mapa de Karnaugh de três variáveis, fazendo-se a simplificação convencional:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



$$F = A \cdot B + A \cdot C + B \cdot C \quad (\text{c.q.d.})$$

Usar mapas de Karnaugh com variáveis inseridas revela-se muito útil nas situações em que se tenham muitas variáveis e se procure a expressão mais simplificada com recurso a mapas mais pequenos.



□ Operação lógica XOR (união exclusiva – soma aritmética módulo 2)

Definição: Operação sobre n variáveis, que só toma o valor 1 quando quando um número ímpar dessas variáveis tomar o valor 1.

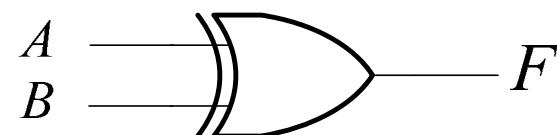
Tabela de verdade

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Expressão algébrica

$$\boxed{F = A \oplus B \\ \equiv \overline{A} \cdot B + A \cdot \overline{B}}$$

Símbolo lógico





□ Operação lógica XOR (união exclusiva – soma aritmética módulo 2)

Propriedades:

$$A \oplus 0 = A \quad \text{Elemento neutro}$$

$$A \oplus 1 = \bar{A} \quad \text{Elemento de complementação}$$

$$A \oplus A = 0$$

$$A \oplus \bar{A} = 1 \quad \text{Tautologia}$$

$$A \oplus B = B \oplus A \quad \text{Propriedade comutativa}$$

$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C \quad \text{Propriedade associativa}$$



□ Operação lógica XNOR (equivalência)

Definição: Operação sobre n variáveis, que só toma o valor 1 quando quando um número par dessas variáveis tomar o valor 1.

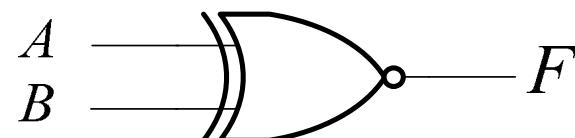
Tabela de verdade

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Expressão algébrica

$$\begin{aligned} F &= \overline{A \oplus B} \\ &\equiv \overline{A} \cdot \overline{B} + A \cdot B \end{aligned}$$

Símbolo lógico



□ Módulo semi-somador (*half-adder*)

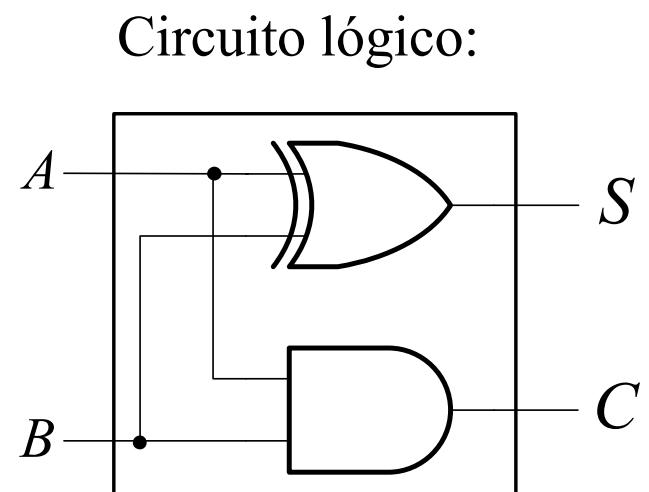
Se se encarar só a soma dos bits dentro de cada peso, obtém-se, para a soma e para o *carry*, a seguinte tabela de verdade:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Expressões algébricas:

$$S = \bar{A} \cdot B + A \cdot \bar{B} = A \oplus B$$

$$C = A \cdot B$$





□ Módulo somador completo (*full-adder*)

O *half-adder* apenas soma dois algarismos de 1 bit, não contando com um possível *carry* que provenha de um peso inferior. Para responder completamente ao algoritmo da soma de dois números binários, é necessário poder somar mais um algarismo (o *carry* do peso imediatamente inferior).

Tabela de verdade do somador completo

C_{n-1}	A_n	B_n	C_n	S_n
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



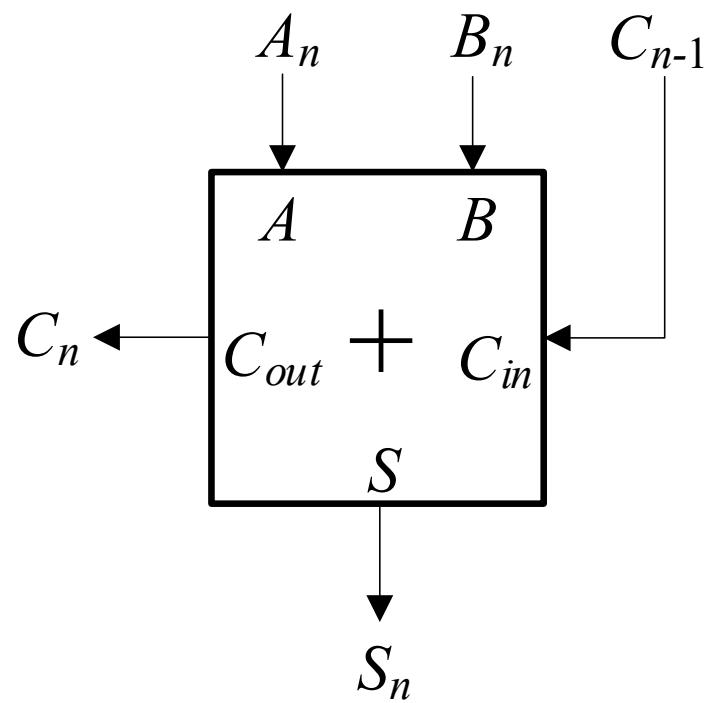
□ Módulo somador completo (*full-adder*)

C_{n-1}	A_n	B_n	C_n	S_n
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S_n = f(A_n, B_n, C_{n-1})$$

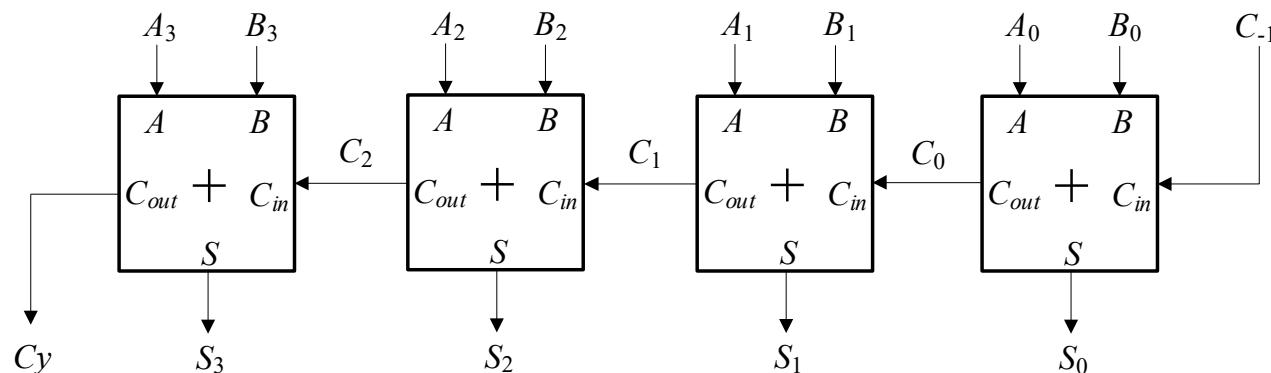
$$C_n = g(A_n, B_n, C_{n-1})$$

Símbolo lógico do somador completo



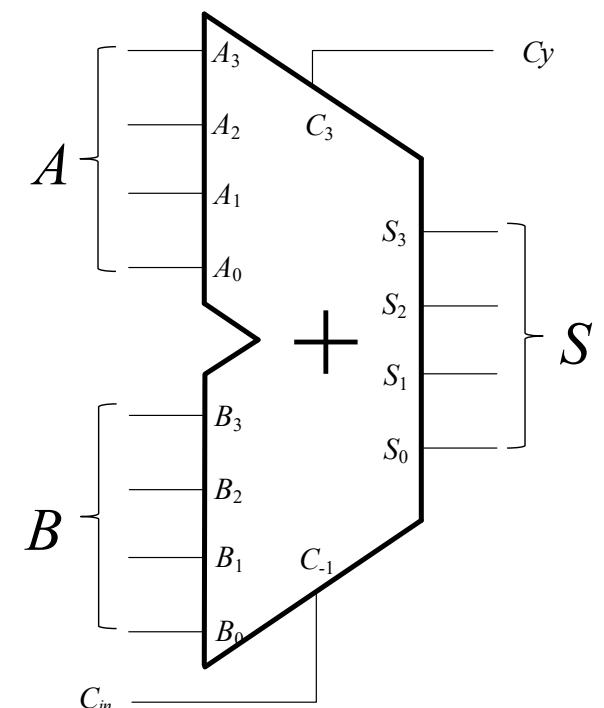
□ Somador de dois números de quatro bits

A base 2 permite um algoritmo, tal que a soma se realiza sobre os bits do mesmo peso, provocando *carry* para o peso seguinte, independentemente da ordem e da dimensão do número.



Concatenação de somadores completos, segundo uma estrutura iterativa.

Símbolo lógico do somador de dois números de quatro bits



$$S = A + B$$



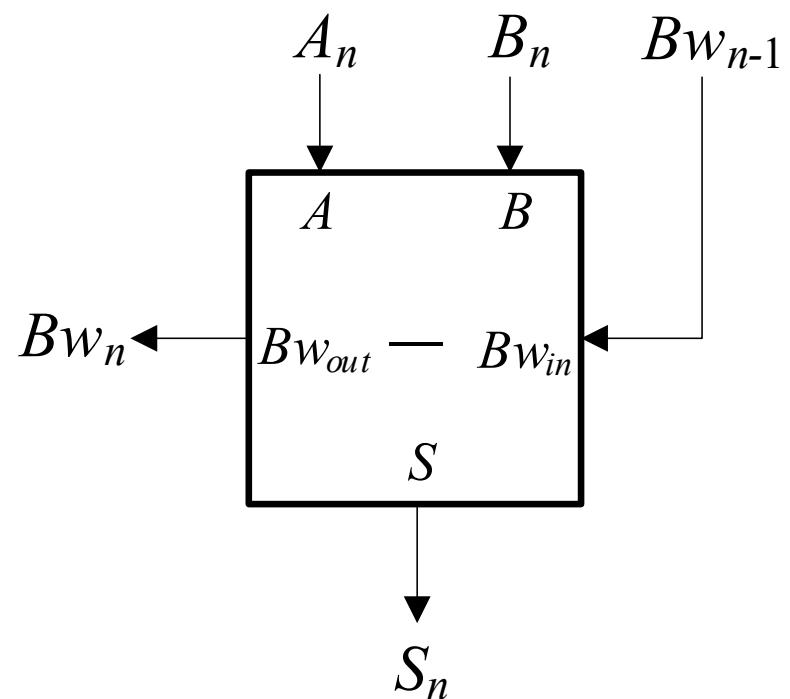
□ Módulo subtrator completo

Bw_{n-1}	A_n	B_n	Bw_n	S_n
0	0	0	?	?
0	0	1	?	?
0	1	0	?	?
0	1	1	?	?
1	0	0	?	?
1	0	1	?	?
1	1	0	?	?
1	1	1	?	?

$$S_n = h(A_n, B_n, Bw_{n-1})$$

$$Bw_n = j(A_n, B_n, Bw_{n-1})$$

Símbolo lógico do subtrator completo



Algoritmo da subtração a 1 bit

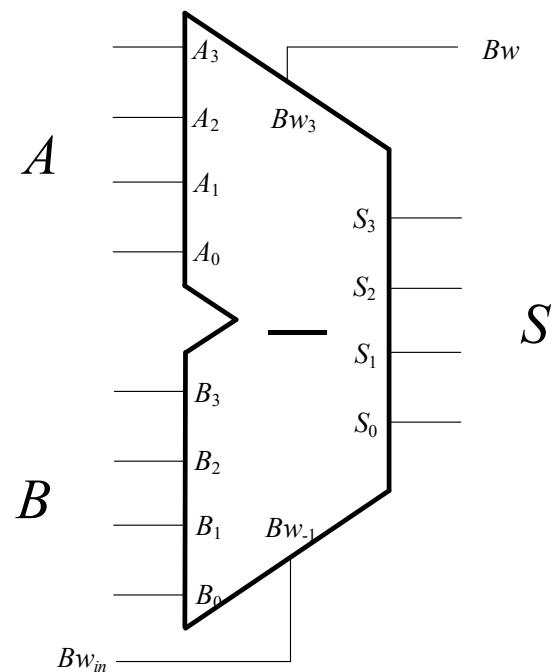


□ Subtração de dois números de quatro bits

A subtração de dois números binários pode ser feita recorrendo à concatenação de vários subtratores completos, de forma semelhante à do somador.

Em cada subtrator completo vai ter-se a operação da subtração direta, provocando um *Borrow* para o peso seguinte

Círcuito subtrator de dois números de quatro bits



$$S = A - B$$



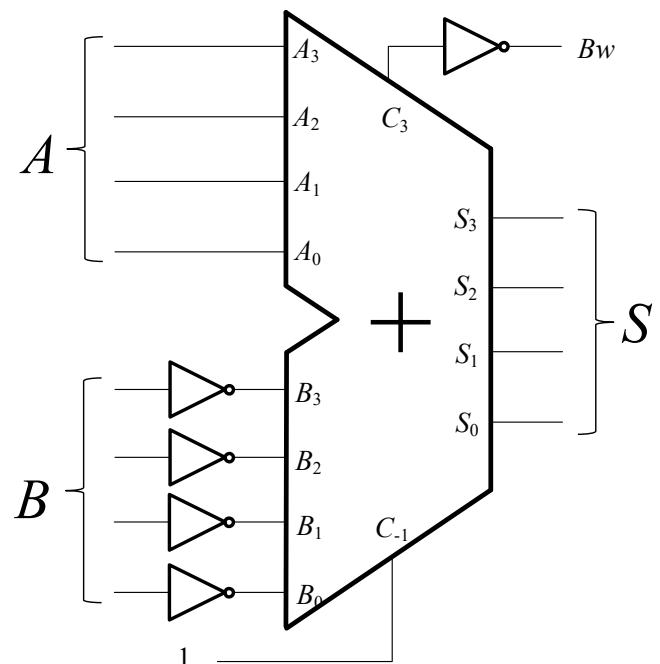
□ Subtração de dois números de quatro bits

Em alternativa, a subtração de dois números binários também pode ser feita recorrendo à soma com o simétrico do subtrativo:

$$S = A - B = A + (-B)$$

Desta forma, pode-se realizar a subtração, recorrendo ao mesmo *hardware* da soma.

Círcuito subtrator de dois números de quatro bits

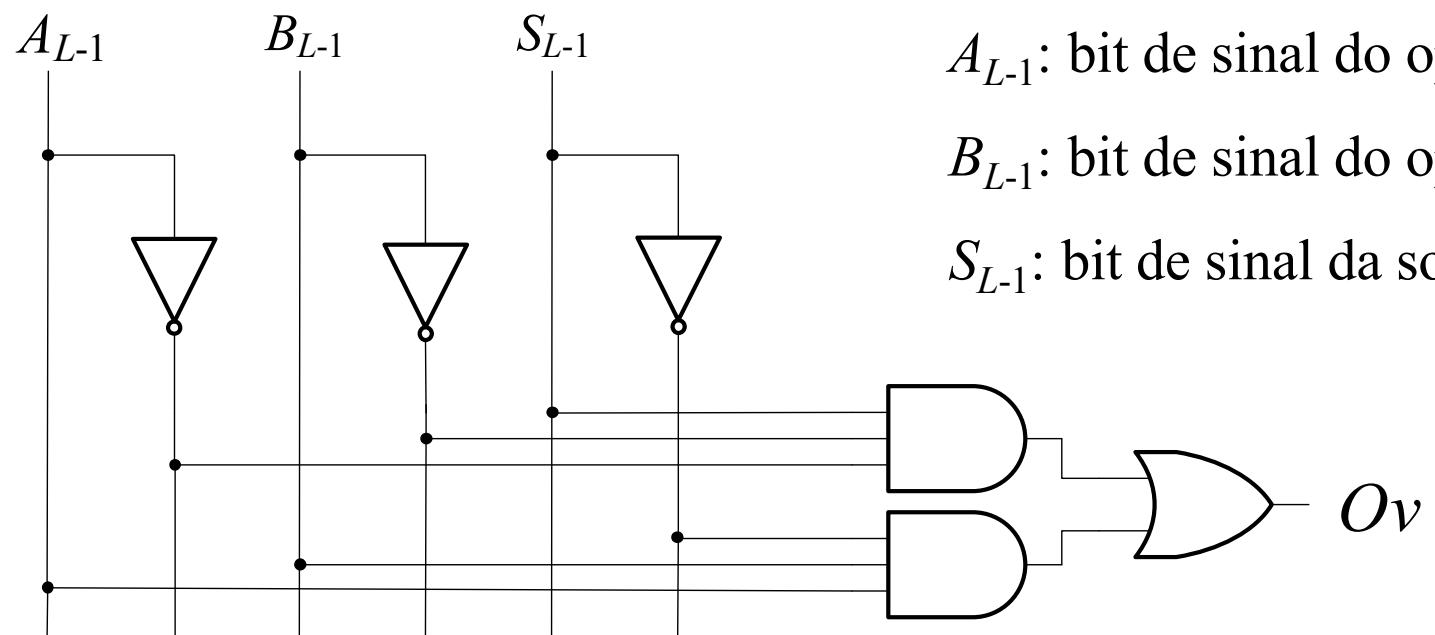


$$S = A - B$$



□ Circuito detetor de *overflow*, pela definição

Por definição, o *overflow* (Ov) ocorre se, tendo como operandos dois números positivos, o resultado da soma for negativo, e vice-versa. O circuito que responde diretamente a esta definição, para números a L bits, é o seguinte:

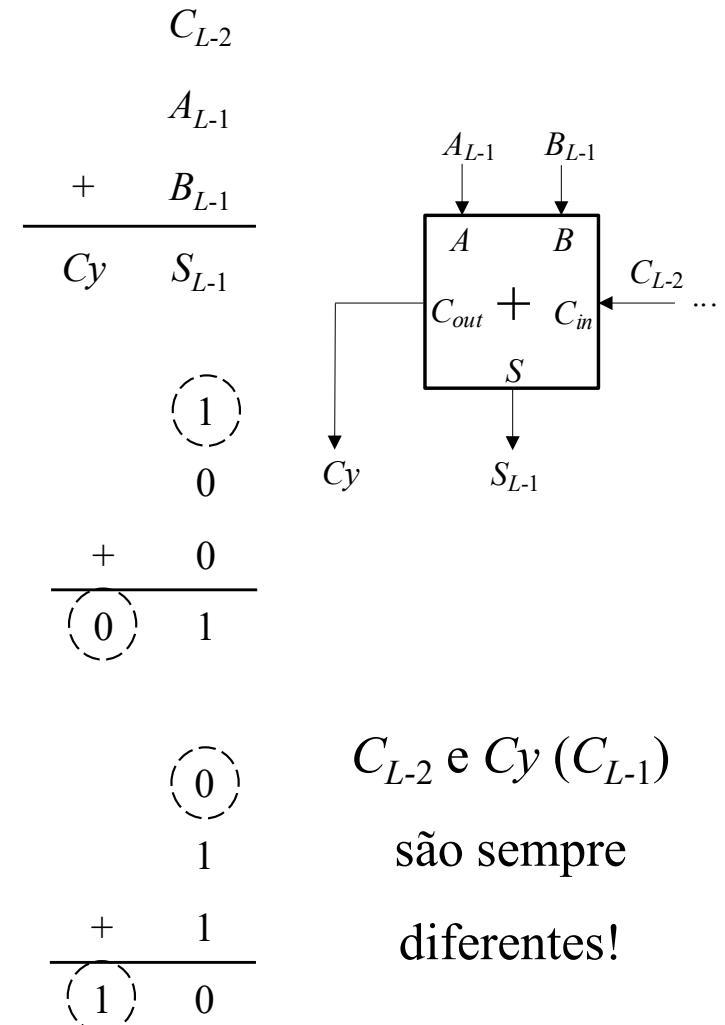




□ Circuito detetor de *overflow*

A situação de *overflow* (*Ov*) ocorre quando, no **módulo de maior peso** do somador completo a L bits:

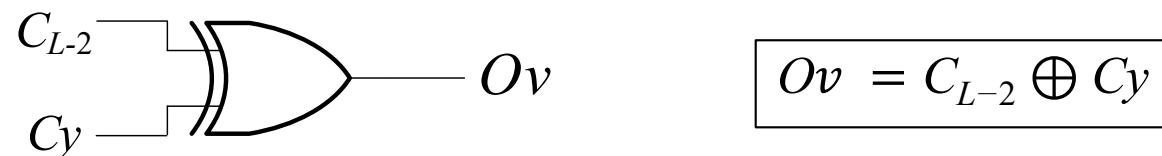
- Os operandos A e B são positivos (bit de sinal é 0) e a soma dá negativa (bit de sinal é 1), e entrou nesse último módulo somador completo um *carry* a 1, não se produzindo *carry* na saída;
- Os operandos, sendo negativos, dão soma positiva e entra 0 no *carry in*, saíndo 1 *carry out*.





□ Operação lógica XOR (aplicação)

Tendo acesso ao bit interno C_{L-2} do módulo *full-adder* de maior peso, a deteção de *overflow* pode ser feita através do XOR entre os bits C_{L-2} e C_y .



Caso o bit C_{L-2} do módulo *full-adder* de maior peso não esteja acessível, pode adotar-se um método diferente. Pode provar-se que $C_{L-2} = A_{L-1} \oplus B_{L-1} \oplus S_{L-1}$.

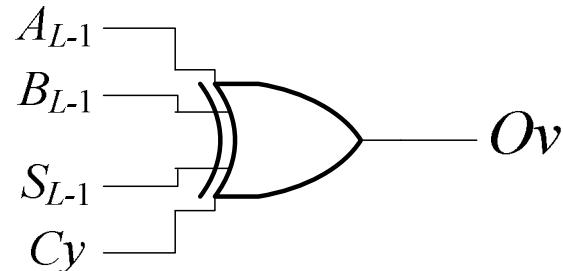
Assim,

$$Ov = A_{L-1} \oplus B_{L-1} \oplus S_{L-1} \oplus C_y$$



□ Operação lógica XOR (aplicação)

Desta forma, num somador com L bits, a deteção de *overflow* pode ser feita recorrendo ao cálculo da função XOR sobre os bits de sinal dos operados que intervêm na soma, o bit de sinal do resultado (pesos $L-1$ para cada um), e o *carry* final resultante da soma.



Exemplo para um somador de quatro bits ($L = 4$):

$$Ov = A_3 \oplus B_3 \oplus S_3 \oplus Cy = 0 \oplus 0 \oplus 1 \oplus 0 = 1$$

$$(\mathbb{N})_{10} \qquad (\mathbb{Z})_{10}$$

A	0	1	1	0	6	(+6)	$Cy = 0$	
$+ B$	+	0	1	1	0	+	(+6)	$Ov = 1$
<hr/>	<hr/>							
S	0	1	1	0	12	-	(-4)	



☐ Variáveis lógicas no Arduino

No Arduino, o tipo de dados `boolean` é o utilizado para conter valores e variáveis lógicas.

- Em essência, as variáveis do tipo `boolean` deveriam ocupar apenas um bit, valendo 0 ou 1. Na realidade, ocupam um byte.
- O valor `true` é qualquer valor diferente de zero. Os símbolos mais utilizados são `true`, `HIGH` ou, simplesmente, `1`.
- O valor `false` é o valor zero. Os símbolos mais utilizados são `false`, `LOW` ou, simplesmente, `0`.



□ Operadores lógicos no Arduino

A linguagem utilizada na programação do Arduino dispõe de vários operadores lógicos que podem usar-se para construir expressões lógicas.

Existem:

- Operadores *booleanos*: usados quando se pretende compôr expressões das quais resulte um valor `true` ou `false`. Dispõem-se do `&&` (AND), `||` (OR) e `!` (NOT).
- Operadores *bitwise* (bit a bit): usados quando se pretende realizar operações lógicas ao nível do bit. Dispõem-se do `&` (AND), `|` (OR), `~` (NOT) e `^` (XOR).



☐ Operadores lógicos no Arduino

- Pode utilizar-se o Arduino para simular as operações lógicas que os circuitos integrados realizariam em *hardware*.
- As variáveis lógicas externas são lidas para uma variável boolean (lógica) através da função `digitalRead(Dx)`, que lê o valor do pino digital (físico).
- Em termos de variedade e de simplicidade de escrita, podem utilizar-se os operadores *bitwise* para realizar as operações lógicas no Arduino, à exceção da operação NOT, em deve utilizar-se o operador que não é *bitwise*.

(AND) A & B

(OR) A | B

(XOR) A ^ B

(NOT) !A



□ Operadores lógicos no Arduino - exemplos

Considere-se variáveis byte $A = B10011001$ e $B = B00111010$

(AND) - $A \& B = 1$ (true)

• Com operadores *boolean*, (OR) - $A | B = 1$ (true)

o resultado fica: (NOT) - $\neg A = 0$ (false)

(AND) - $A \& B = 00011000$ (true)

• Com operadores *bitwise*, (OR) - $A | B = 10111011$ (true)

o resultado fica: (XOR) - $A ^ B = 10100011$ (true)

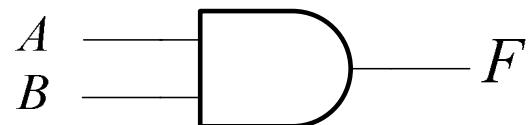
(NOT) - $\neg A = 01100110$ (true)



□ Operação lógica AND no Arduino

Tendo em conta a definição feita anteriormente, ou seja:

$$F = A \cdot B$$



Pode usar-se o operador `&` diretamente, ou definir-se uma função AND de duas variáveis da seguinte forma, em que o valor de retorno e as variáveis de entrada, sendo booleanos, só poderá resultar em *true* ou *false*.

```
boolean AND (boolean A, boolean B) {  
    return A & B;  
}
```



□ Operação lógica AND no Arduino - exemplos e variantes

Considerando, por exemplo, $A = B10011001$ e $B = B00111010$, para a função anterior:

```
boolean AND (boolean A, boolean B) {  
    return A & B;  
}
```

Evocando-se $F = \text{AND}(A, B)$ resultará em $F = 1$ (true).

Alterando os tipos de dados e redefinindo a função para:

```
byte AND (byte A, byte B) {  
    return A & B;  
}
```

Evocando-se $F = \text{AND}(A, B)$ resultará em $F = 00011000$.



□ Operação lógica AND no Arduino - exemplos e variantes

Considerando-se a definição anterior,

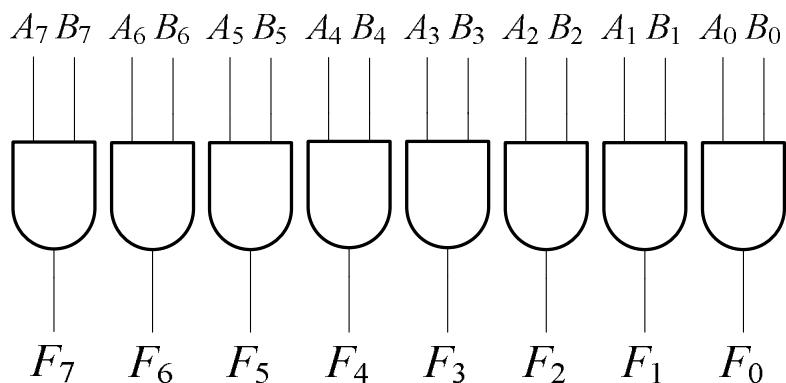
```
byte AND (byte A, byte B) {  
    return A & B;  
}
```

A e B correspondem a $A = A_7A_6A_5A_4A_3A_2A_1A_0$ e $B = B_7B_6B_5B_4B_3B_2B_1B_0$.

Tendo-se $F = \text{AND}(A, B)$, F corresponderá a $F = F_7F_6F_5F_4F_3F_2F_1F_0$.

Se $A = B10011001$ e $B = B00111010$, $F = B00011000$.

A função operará como se do seguinte circuito se tratasse:

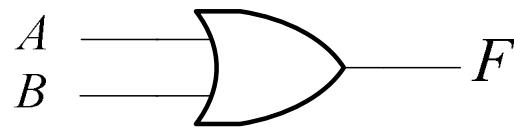




☐ Operação lógica OR no Arduino

Tendo em conta a definição feita anteriormente, ou seja:

$$F = A + B$$



Pode usar-se o operador `|` diretamente, ou definir-se uma função OR de duas variáveis da seguinte forma, em que o valor de retorno e as variáveis de entrada, sendo booleanos, só poderá resultar em *true* ou *false*.

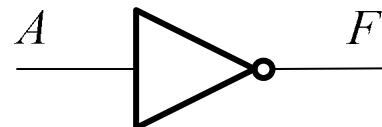
```
boolean OR (boolean A, boolean B) {  
    return A | B;  
}
```



☐ Operação lógica NOT no Arduino

Tendo em conta a definição feita anteriormente, ou seja:

$$F = \overline{A}$$



Pode usar-se o operador `!` diretamente, ou definir-se uma função NOT de uma variável da seguinte forma, em que o valor de retorno e a variável de entrada, sendo booleanos, só poderá resultar em *true* ou *false*.

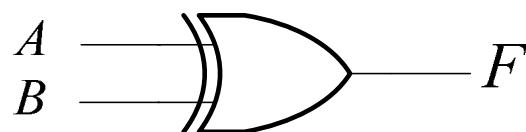
```
boolean NOT (boolean A) {  
    return !A;  
}
```



□ Operação lógica XOR no Arduino

Tendo em conta a definição feita anteriormente, ou seja:

$$F = A \oplus B$$



Pode usar-se o operador `^` diretamente, ou definir-se uma função XOR de duas variáveis da seguinte forma, em que o valor de retorno e as variáveis de entrada, sendo booleanos, só poderá resultar em *true* ou *false*.

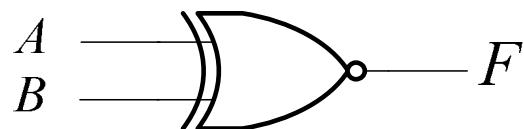
```
boolean XOR (boolean A, boolean B) {  
    return A ^ B;  
}
```



□ Operação lógica XNOR no Arduino

Tendo em conta a definição feita anteriormente, ou seja:

$$F = \overline{A \oplus B}$$



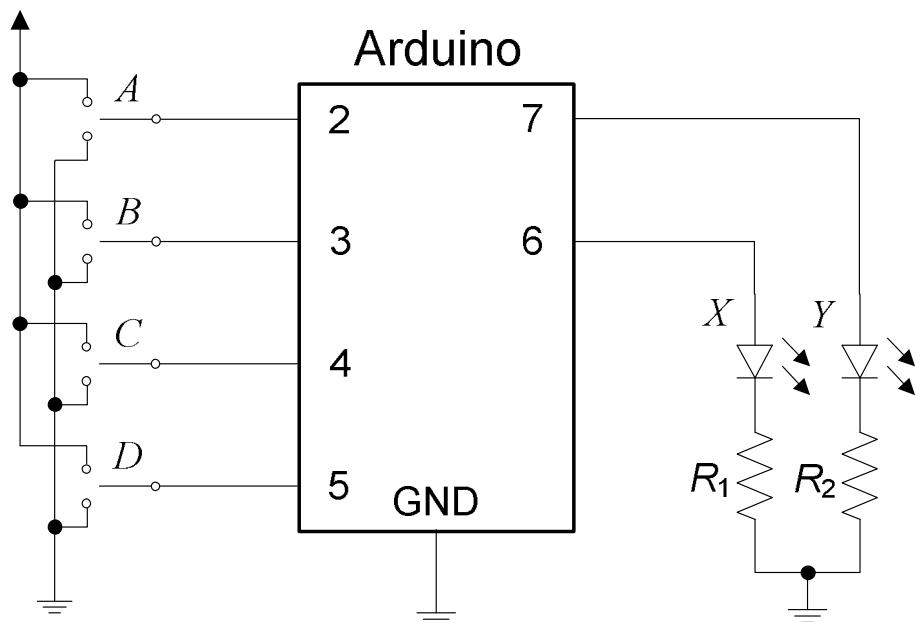
Podem usar-se os operadores ! e ^ diretamente, ou definir-se uma função XNOR de duas variáveis da seguinte forma, em que o valor de retorno e as variáveis de entrada, sendo booleanos, só poderá resultar em *true* ou *false*.

```
boolean XNOR (boolean A, boolean B) {  
    return !(A ^ B);  
}
```



☐ Exemplo de implementação das funções de maioria no Arduino

Esquema elétrico do circuito:



As entradas A , B , C e D têm de estar ligadas a um dos dois terminais (5 V e GND), de maneira a estabelecer o valor lógico das variáveis.

As saídas X e Y são constituídas, cada uma, por um LED e uma resistência em série.



☐ Exemplo de implementação das funções de maioria no Arduino (cont.)

$$X = F_1(A, B, C, D) = A.C.D + A.B.D + B.C.D + A.B.C$$

$$Y = F_2(A, B, C, D) = \overline{A}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{D} + \overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}$$

```
// Atribuição de pinos de entrada (A, B, C, D) e de saída (X, Y)
```

```
#define pinA 2
#define pinB 3
#define pinC 4
#define pinD 5
#define pinX 6
#define pinY 7
```

```
// Variáveis lógicas de entrada e de saída (globais)
```

```
boolean A, B, C, D, X, Y;
```

```
// Função lógica de maioria de "1"s
```

```
boolean F1(boolean A, boolean B, boolean C, boolean D) {
    return A & C & D | A & B & D | B & C & D | A & B & C;
}
```

```
// Função lógica de maioria de "0"s
```

```
boolean F2(boolean A, boolean B, boolean C, boolean D) {
    return !A & !C & !D | !A & !B & !D | !B & !C & !D | !A & !B & !C;
}
```



☐ Exemplo de implementação das funções de maioria no Arduino (cont.)

$$X = F_1(A, B, C, D) = A.C.D + A.B.D + B.C.D + A.B.C$$

$$Y = F_2(A, B, C, D) = \overline{A}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{D} + \overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}$$

```
// Leitura de pinos físicos digitais para as variáveis de entrada
void lerEntradas() {
    A = digitalRead(pinA);
    B = digitalRead(pinB);
    C = digitalRead(pinC);
    D = digitalRead(pinD);
}

// Calcular o valor das funções de maioria
void calcularFuncoesCombinatorias() {
    X = F1(A, B, C, D);
    Y = F2(A, B, C, D);
}

// Escrita das variáveis de saída em pinos físicos digitais
void escreverSaidas() {
    digitalWrite(pinX, X);
    digitalWrite(pinY, Y);
}
```



☐ Exemplo de implementação das funções de maioria no Arduino (cont.)

$$X = F_1(A, B, C, D) = A.C.D + A.B.D + B.C.D + A.B.C$$

$$Y = F_2(A, B, C, D) = \overline{A}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{D} + \overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}$$

```
// Configuração dos pinos de saída e de entrada
```

```
void setup() {
    pinMode(pinA, INPUT);
    pinMode(pinB, INPUT);
    pinMode(pinC, INPUT);
    pinMode(pinD, INPUT);
    pinMode(pinX, OUTPUT);
    pinMode(pinY, OUTPUT);
}
```

```
// Execução cíclica das funções lógicas
```

```
void loop() {
    lerEntradas();
    calcularFuncoesCombinatorias();
    escreverSaidas();
}
```



☐ Exemplo de implementação das funções de maioria no Arduino (cont.)

$$X = F_1(A, B, C, D) = A.C.D + A.B.D + B.C.D + A.B.C$$
$$Y = F_2(A, B, C, D) = \overline{A}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{D} + \overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}$$

```
// Função lógica de maioria de "1"s
boolean F1(boolean A, boolean B, boolean C, boolean D){
    return A & C & D | A & B & D | B & C & D | A & B & C;
}

// Função lógica de maioria de "0"s
boolean F2(boolean A, boolean B, boolean C, boolean D){
    return !A & !C & !D | !A & !B & !D | !B & !C & !D | !A & !B & !C;
}
// Calcular o valor das funções de maioria (Versão mais compacta)
void calcularFuncoesCombinatorias(){
    digitalWrite(pinX, F1(digitalRead(pinA), digitalRead(pinB), digitalRead(pinC), digitalRead(pinD)));
    digitalWrite(pinY, F2(digitalRead(pinA), digitalRead(pinB), digitalRead(pinC), digitalRead(pinD)));
}

// Configuração dos pinos de saída (os outros, por default, são entrada)
void setup(){
    pinMode(pinX, OUTPUT);
    pinMode(pinY, OUTPUT);
}

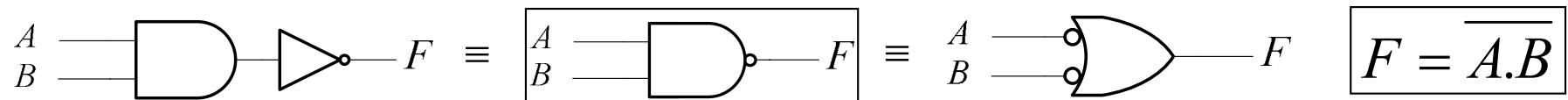
// Execução cíclica das funções lógicas
void loop(){
    calcularFuncoesCombinatorias();
}
```

Esta versão não precisa de declaração de variáveis, dado que se têm em atenção apenas os pinos digitais de entrada.



□ Operação lógica NAND (aglutinação das funções NOT e AND)

Definição: Operação sobre n variáveis, que só toma o valor 0 quando todas essas variáveis tiverem o valor 1.



Propriedades:

$$\begin{aligned} A \uparrow 1 &= \overline{A \cdot 1} = \overline{A} \\ A \uparrow 0 &= \overline{A \cdot 0} = 1 \\ A \uparrow A &= \overline{A \cdot A} = \overline{A} \\ A \uparrow \overline{A} &= \overline{A \cdot \overline{A}} = 1 \\ A \uparrow B &= B \uparrow A \\ (A \uparrow B) \uparrow C &= \overline{\overline{A \cdot B} \cdot C} = A \cdot B + \overline{C} = \overline{\overline{A} + \overline{B} + \overline{C}} \\ A \uparrow B \uparrow C &= \overline{\overline{A \cdot B} \cdot C} = \overline{A} + \overline{B} + \overline{C} \neq (A \uparrow B) \uparrow C \end{aligned}$$

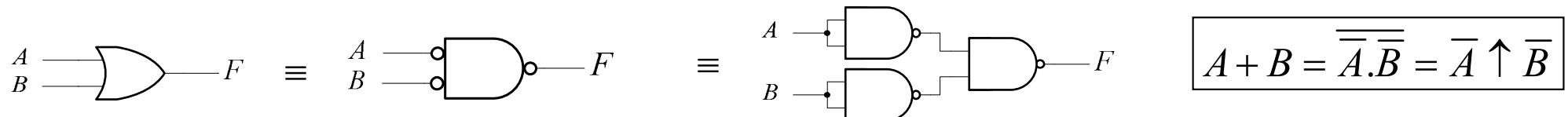
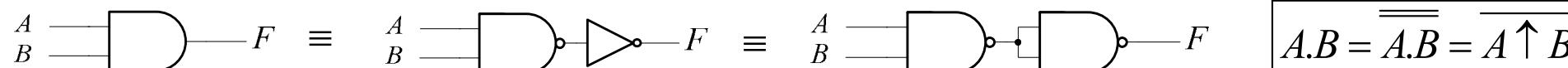
- Comutativa
- Não associativa

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0



□ Operação lógica NAND (aglutinação das funções NOT e AND)

A função NAND diz-se funcionalmente completa porque, a partir dela, podem sintetizar-se as funções lógicas básicas NOT, AND e OR.



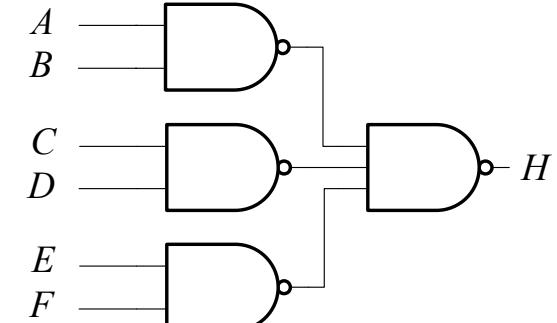
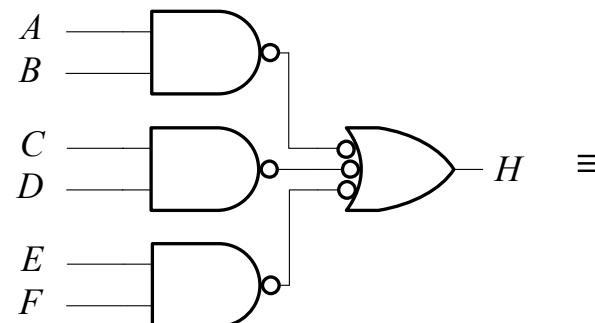
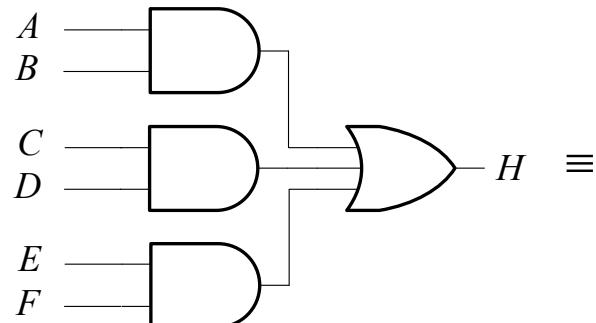
Assim, qualquer função lógica pode ser realizada usando exclusivamente portas lógicas do tipo NAND.

□ Operação lógica NAND (aglutinação das funções NOT e AND)

As expressões AND-OR têm imediata conversão para portas NAND.

Exemplo:

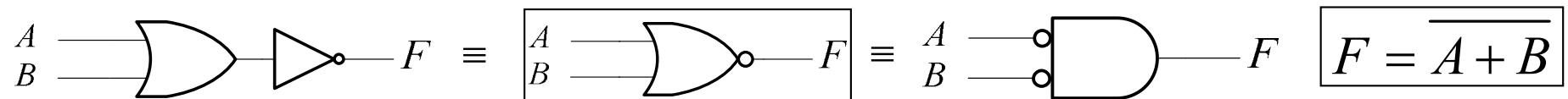
$$H = A \cdot B + C \cdot D + E \cdot F = \overline{\overline{A} \cdot \overline{B}} + \overline{\overline{C} \cdot \overline{D}} + \overline{\overline{E} \cdot \overline{F}} = \overline{\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \cdot \overline{E} \cdot \overline{F}} = (A \uparrow B) \uparrow (C \uparrow D) \uparrow (E \uparrow F)$$





□ Operação lógica NOR (aglutinação das funções NOT e OR)

Definição: Operação sobre n variáveis, que só toma o valor 1 quando todas essas variáveis tiverem o valor 0.



Propriedades:

$$A \downarrow 0 = \overline{A + 0} = \overline{A}$$

• Comutativa

$$A \downarrow 1 = \overline{A + 1} = 0$$

• Não associativa

$$A \downarrow A = \overline{A + A} = \overline{A}$$

$$A \downarrow \overline{A} = \overline{A + \overline{A}} = 0$$

$$A \downarrow B = B \downarrow A$$

$$(A \downarrow B) \downarrow C = \overline{\overline{A + B} + C} = (A + B). \overline{C} = \overline{A}. \overline{B}. \overline{C}$$

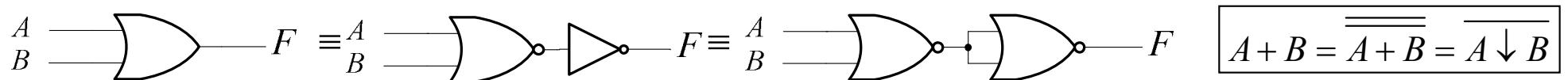
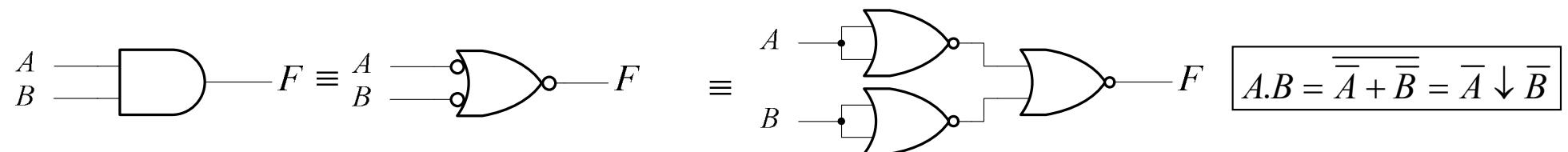
$$A \downarrow B \downarrow C = \overline{A + B + C} = \overline{A}. \overline{B}. \overline{C} \neq (A \downarrow B) \downarrow C$$

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0



□ Operação lógica NOR (aglutinação das funções NOT e OR)

A função NOR diz-se funcionalmente completa porque, a partir dela, podem sintetizar-se as funções lógicas básicas NOT, AND e OR.



Assim, qualquer função lógica pode ser realizada usando exclusivamente portas lógicas do tipo NOR.

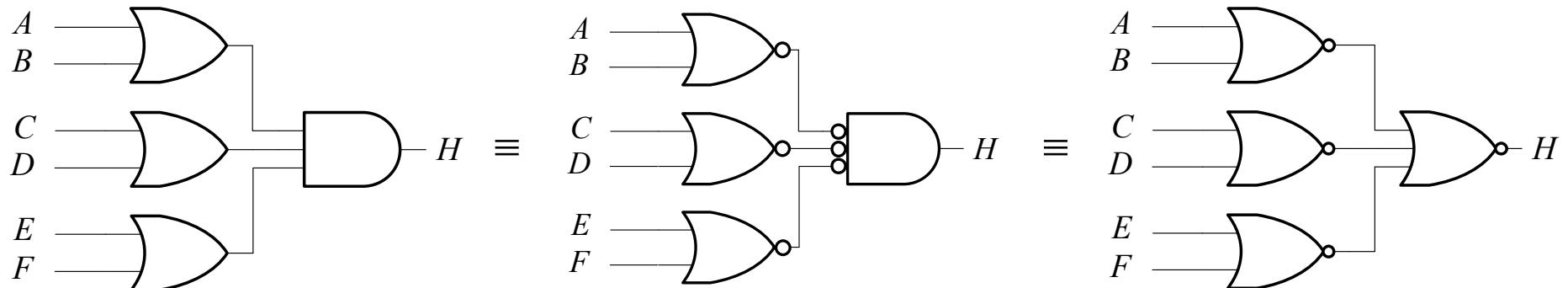


□ Operação lógica NOR (aglutinação das funções NOT e OR)

As expressões OR-AND têm imediata conversão para portas NOR.

Exemplo:

$$H = (A + B).(C + D).(E + F) = \overline{\overline{A} + \overline{B}} \cdot \overline{\overline{C} + \overline{D}} \cdot \overline{\overline{E} + \overline{F}} = \\ = \overline{\overline{A + B} + \overline{C + D} + \overline{E + F}} = (A \downarrow B) \downarrow (C \downarrow D) \downarrow (E \downarrow F)$$

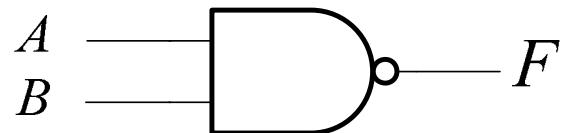




☐ Operação lógica NAND no Arduino

Tendo em conta a definição feita anteriormente, ou seja:

$$F = \overline{A \cdot B}$$



Podem usar-se os operadores ! e & diretamente, ou definir-se uma função NAND de duas variáveis da seguinte forma, em que o valor de retorno e as variáveis de entrada, sendo booleanos, só poderá resultar em *true* ou *false*.

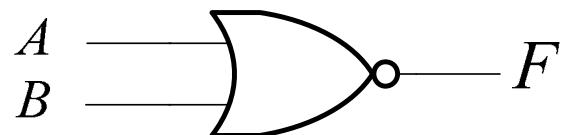
```
boolean NAND (boolean A, boolean B) {  
    return !(A & B);  
}
```



□ Operação lógica NOR no Arduino

Tendo em conta a definição feita anteriormente, ou seja:

$$F = \overline{A + B}$$



Podem usar-se os operadores `!` e `|` diretamente, ou definir-se uma função NOR de duas variáveis da seguinte forma, em que o valor de retorno e as variáveis de entrada, sendo booleanos, só poderá resultar em *true* ou *false*.

```
boolean NOR (boolean A, boolean B) {  
    return !(A | B);  
}
```



□ Resumo das várias operações (portas) lógicas

Nome da operação	Símbolo lógico	Expressão lógica
NOT		$F = \overline{A}$
AND		$F = A.B$
OR		$F = A + B$
XOR		$F = A \oplus B$
NAND		$F = \overline{A.B}$
NOR		$F = \overline{A + B}$
XNOR		$F = \overline{A \oplus B}$

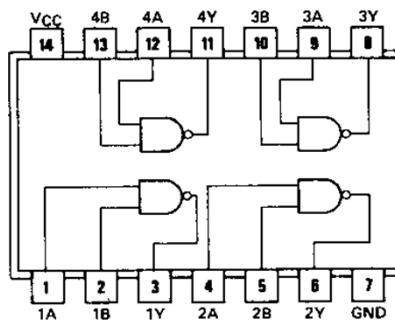


□ Circuitos integrados

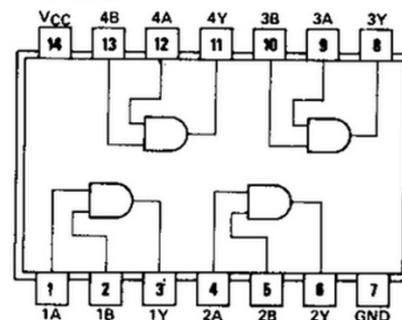
As portas lógicas já estudadas existem comercialmente em circuito integrado, em *chips* que variam na quantidade de portas e no número de entradas por porta.



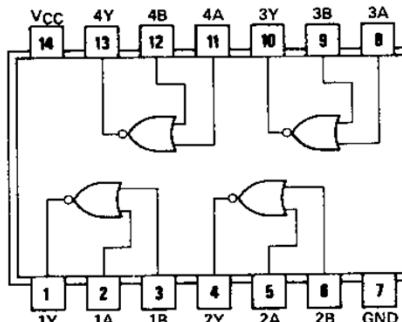
7400



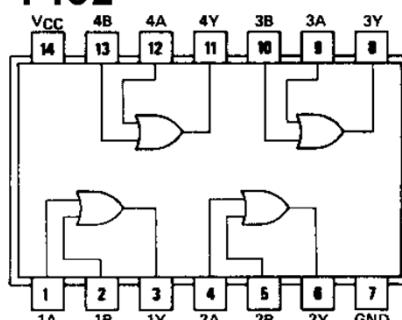
7408



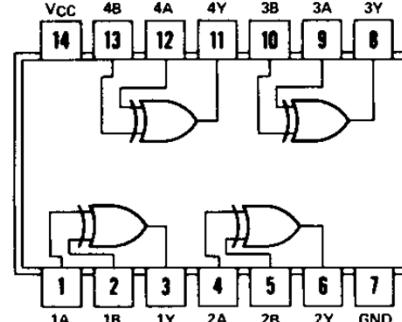
7402



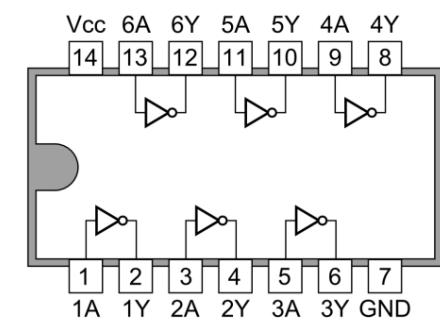
7432



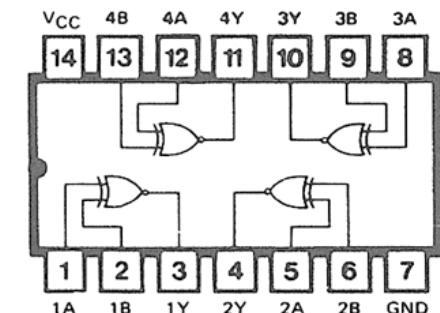
7486



7404 Hex Inverters

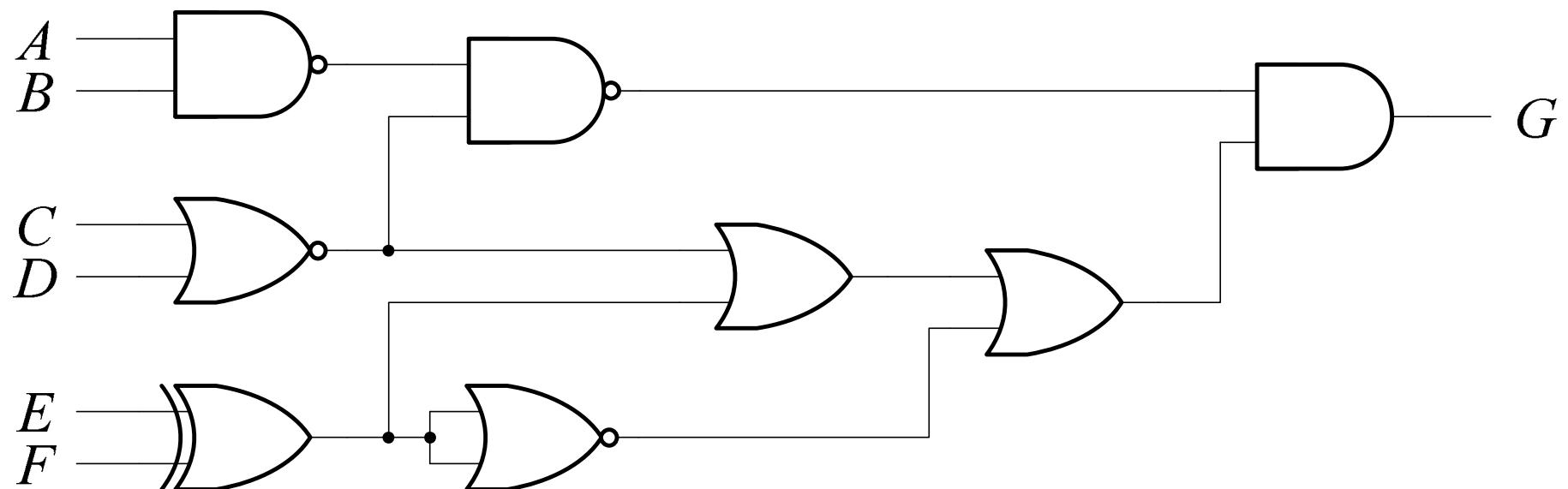


74266



□ Exercício

Obter a expressão simplificada da função G (sugestão: comece do lado das entradas e vá obtendo a expressão lógica à saída de cada porta, simplificando sucessivamente, até chegar à saída).



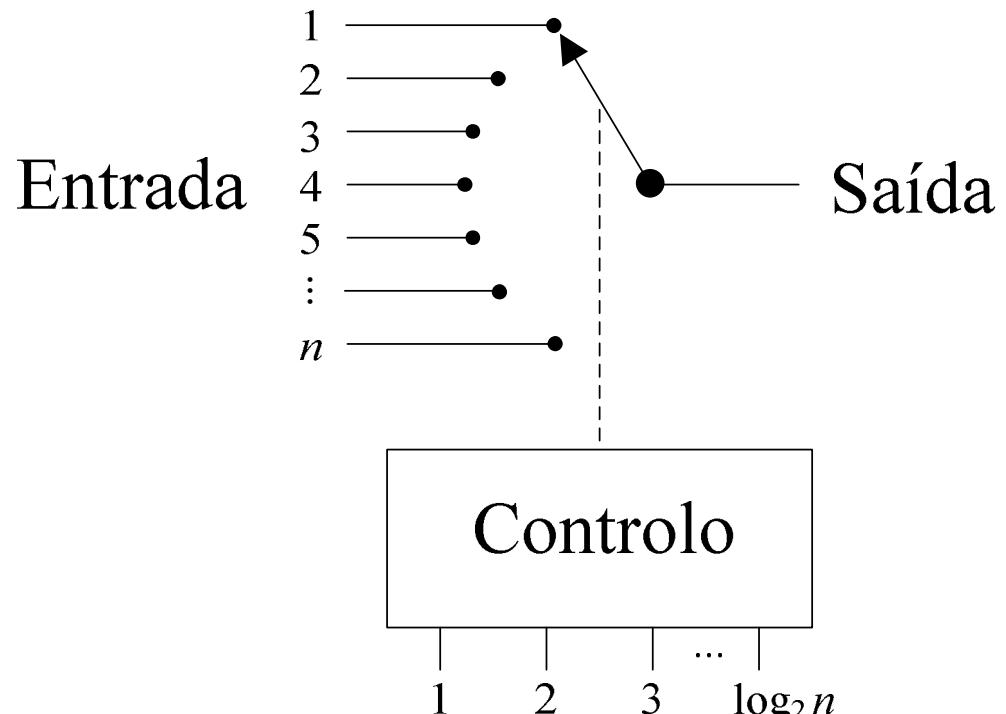


□ Conceito funcional da multiplexagem

O conceito de multiplexagem consiste em comutar (escolher) apenas uma entre n entradas possíveis e ligá-la à saída.

O bloco de controlo, em função dos seus $\log_2 n$ bits, promove a seleção da entrada a ligar à saída.

Um sistema com esta funcionalidade denomina-se multiplexador (*multiplexer*).

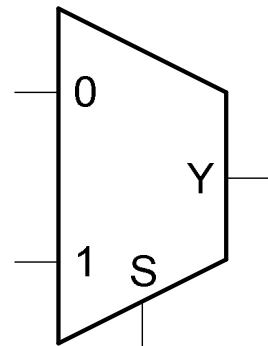




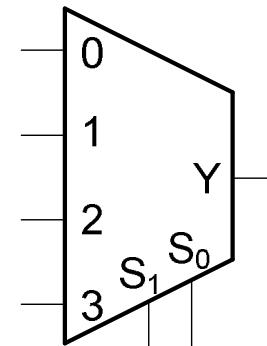
□ Multiplexer lógico

O *multiplexer* lógico assenta no mesmo princípio já descrito genericamente para os *multiplexers*, ou seja, coloca na sua saída o valor lógico presente na entrada determinada pelos bits de seleção.

Símbolos lógicos:



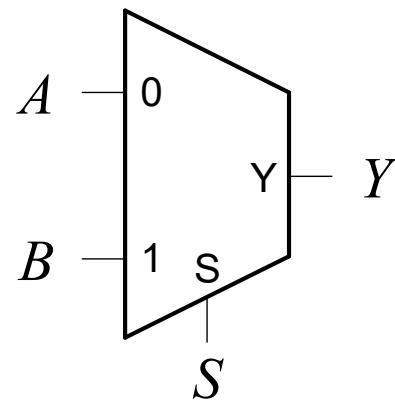
MUX 2×1



MUX 4×1

□ Síntese do *multiplexer* (MUX) 2×1

Trata-se de um circuito com três entradas digitais, logo, cuja saída é determinada pela combinação de três variáveis lógicas.



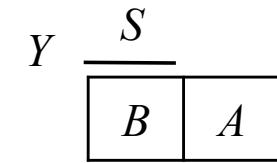
MUX 2×1

$$S = 0: Y = A$$

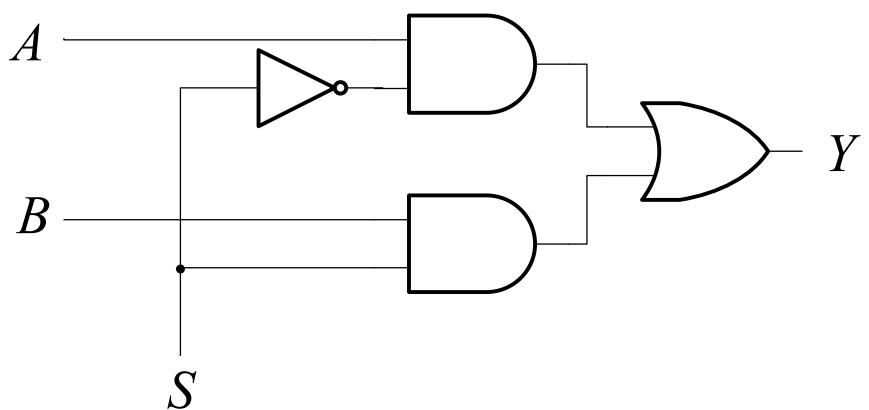
$$S = 1: Y = B$$

S	A	B	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Y	A		
S	0	1	0
	0	0	1
A	1	1	0
B	0	0	1

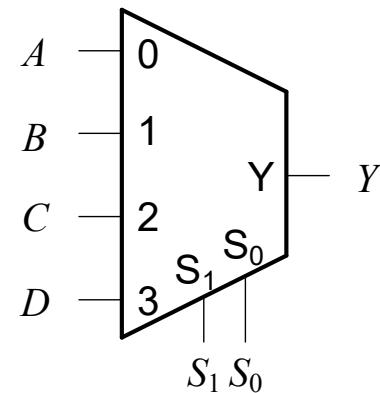


$$Y = \bar{S}.A + S.B$$



□ Circuito do multiplexer (MUX) 4×1

Neste caso, a síntese da função é feita de acordo com uma função de seis variáveis lógicas de entrada.

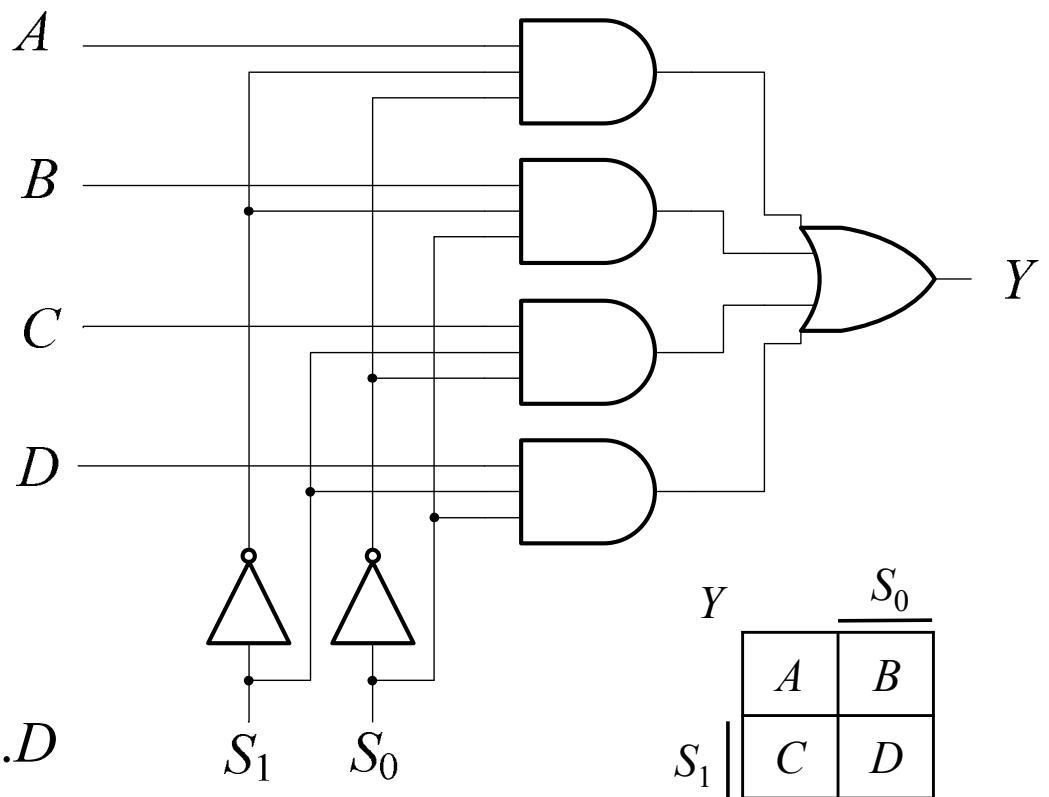


MUX 4×1

S_1	S_0	Y
0	0	A
0	1	B
1	0	C
1	1	D

Tabela de verdade com variáveis inseridas

$$Y = \overline{S_1} \cdot \overline{S_0} \cdot A + \overline{S_1} \cdot S_0 \cdot B + S_1 \cdot \overline{S_0} \cdot C + S_1 \cdot S_0 \cdot D$$



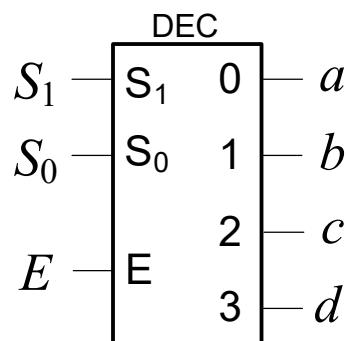
S_1	S_0	Y
0	0	A
0	1	B

S_1	S_0	Y
1	0	C
1	1	D

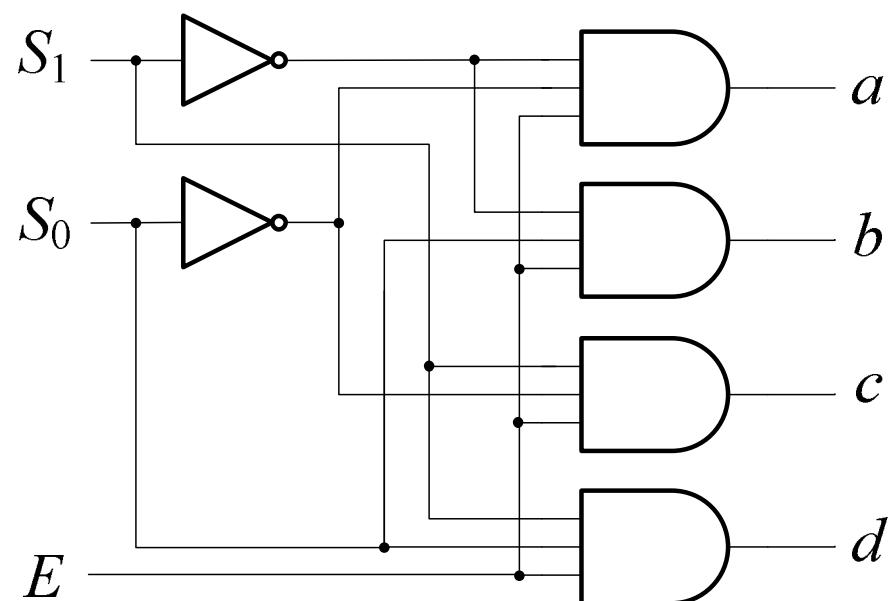


□ Descodificador (decoder)

O *decoder* ativa uma e só uma saída. Este poderá também ter uma entrada de *enable*, a qual permitirá o funcionamento normal do dispositivo. Se o *enable* for igual a “0”, todas as suas saídas deverão ficar a “0”. Com esta definição, o circuito lógico do *decoder* será:



$$\begin{aligned}a &= \overline{S_1} \cdot \overline{S_0} \cdot E \\b &= \overline{S_1} \cdot S_0 \cdot E \\c &= S_1 \cdot \overline{S_0} \cdot E \\d &= S_1 \cdot S_0 \cdot E\end{aligned}$$





Conceito de *enable*

Para consolidar:

Em situações nas quais se tenha a concatenação de vários módulos, pode ser útil colocar inativo qualquer um deles, consoante o objetivo pretendido.

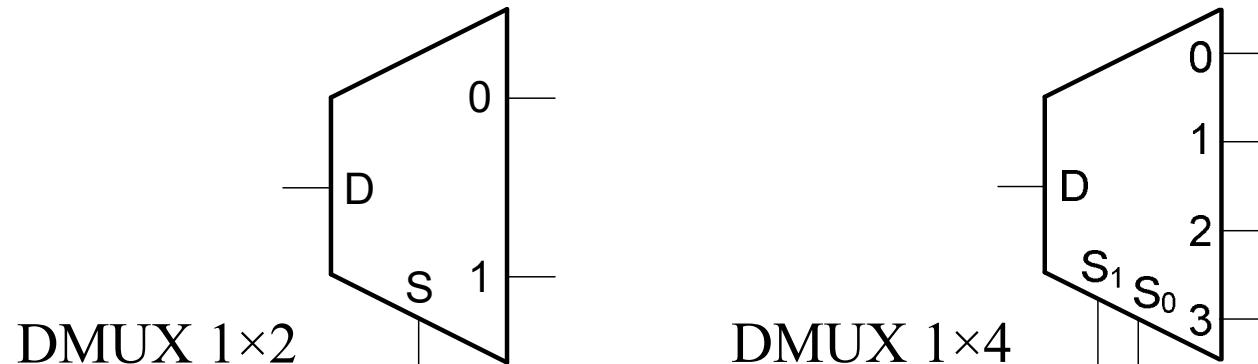
Nestas condições, evidencia-se o sinal de *enable*, o qual quando ativo (pode sê-lo a “1” ou a “0”), permite o funcionamento do respetivo dispositivo em conjunto com os outros aos quais se encontra ligado, ao passo que quando o *enable* está inativo, o mesmo módulo encontrar-se-á inibido de realizar as suas funções.



□ Demultiplexer

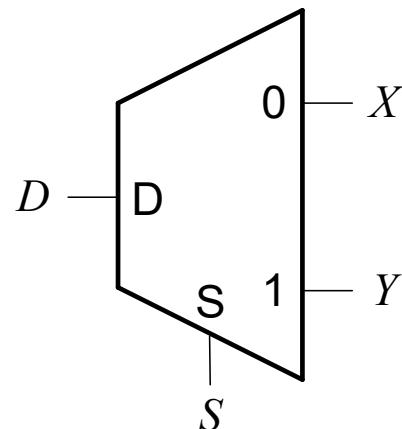
O *demultiplexer* realiza a função inversa do *multiplexer*, ou seja, tendo uma entrada, coloca o valor lógico que esta apresenta numa das suas 2^n saídas, sendo que essa saída é selecionada por n entradas de seleção. A sua função é praticamente igual à do descodificador, diferindo no facto deste último colocar sempre “1” na saída pretendida, ao passo que o *demultiplexer* coloca “0” ou “1” dependendo do valor lógico da entrada de informação.

Símbolos
lógicos:



□ Síntese do *demultiplexer* (DMUX) 1×2

Trata-se de um circuito com duas entradas digitais e com duas saídas, que resultam da combinação das duas variáveis lógicas de entrada.



DMUX 1×2

S	D	X	Y
0	0	0	0
0	1	1	0
1	0	0	0
1	1	0	1

$$S = 0 : X = D; \quad Y = 0$$

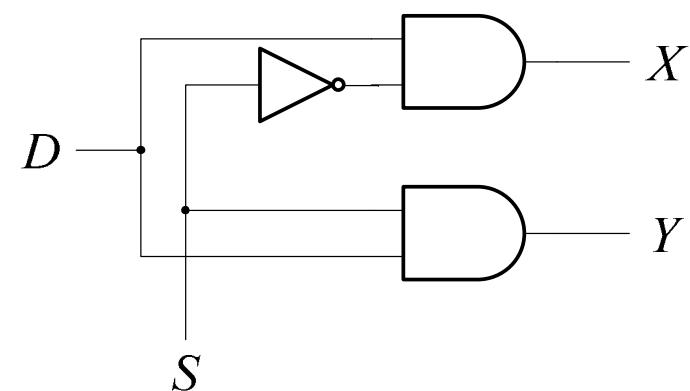
$$S = 1 : X = 0; \quad Y = D$$

X	D
1	0
0	0

$$X = \bar{S}.D$$

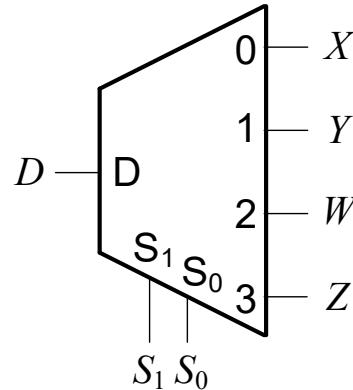
Y	D
0	0
1	0

$$Y = S.D$$



☐ Circuito do demultiplexer (DMUX) 1×4

Neste caso, a síntese de cada uma das quatro funções de saída é feita de acordo com uma função de três variáveis lógicas de entrada.



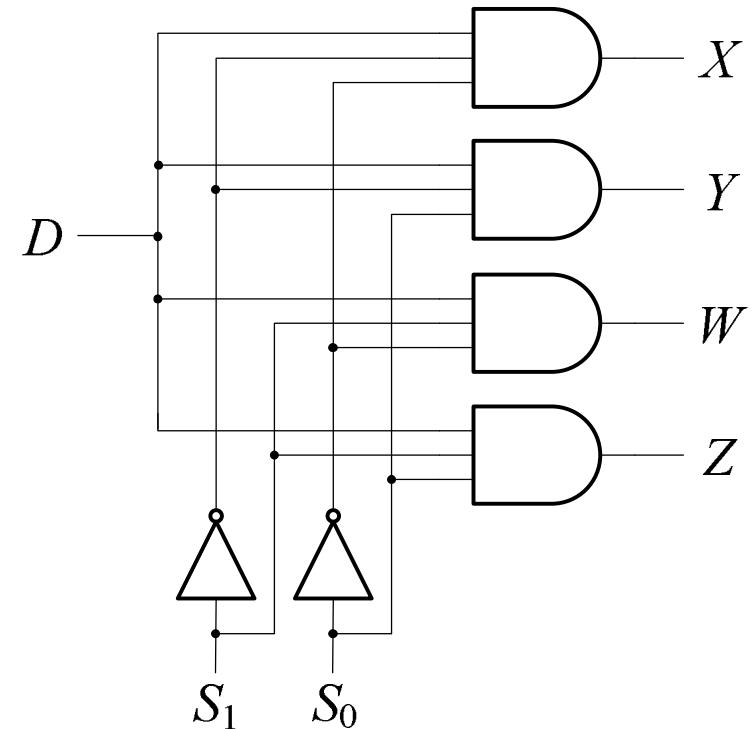
$$X = \overline{S_1} \cdot \overline{S_0} \cdot D$$

$$Y = \overline{S_1} \cdot S_0 \cdot D$$

$$W = S_1 \cdot \overline{S_0} \cdot D$$

$$Z = S_1 \cdot S_0 \cdot D$$

	S_1	S_0	D	X	Y	W	Z
	0	0	0	0	0	0	0
	0	0	1	1	0	0	0
	0	1	0	0	0	0	0
	0	1	1	0	1	0	0
	1	0	0	0	0	0	0
	1	0	1	0	0	1	0
	1	1	0	0	0	0	0
	1	1	1	0	0	0	1

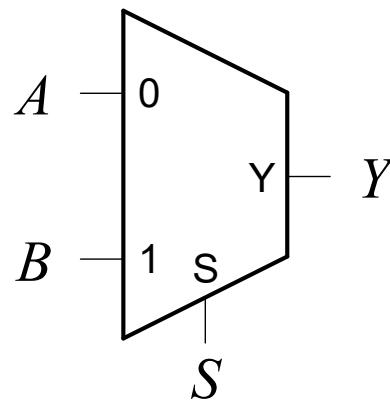




□ Multiplexer no Arduino

A implementação de *multiplexers* no Arduino pode ser feita segundo os mesmos princípios utilizados anteriormente para as outras funções lógicas.

Num MUX que opera só com variáveis do tipo boolean (bits simples) a função de simulação no Arduino pode ser segundo a definição dada anteriormente. Exemplo para o MUX 2×1 (extensível para o MUX 4×1):



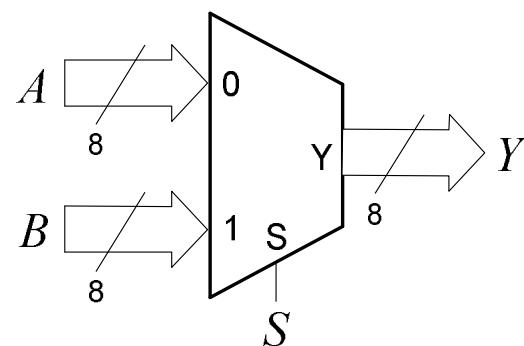
MUX 2×1

```
boolean MUX_2x1 (boolean S, boolean A, boolean B) {  
    return !S & A | S & B;  
}
```



□ Multiplexer no Arduino

Estes módulos são compostos, estão acima do nível da porta lógica. Se se pretender realizar um MUX que tem como entrada variáveis do tipo byte em vez de variáveis boolean (bits), o mais importante é que a função de simulação no Arduino opere segundo o princípio funcional pretendido, não tendo a preocupação de operar ao nível da porta lógica. Assim, pode ficar-se com:



MUX 2×1

```
byte MUX_2x1 (boolean S, byte A, byte B) {  
    return S ? B : A;  
}
```



□ Multiplexer no Arduino

Para o caso de um MUX 4×1 que opere com variáveis do tipo byte, a função que o simula poderá ser:

```
byte MUX_4x1 (boolean S1, boolean S0, byte A, byte B, byte C, byte D) {  
    switch (S1 << 1 | S0) {  
        case B00:  
            return A;  
        case B01:  
            return B;  
        case B10:  
            return C;  
        case B11:  
            return D;  
    }  
}
```

Pela definição de número, e de forma equivalente, a composição do valor do seletor da entrada poderia ser dada por $S1 * 2 + S0$, em vez de $S1 << 1 | S0$



□ Operadores “>>”e “<<” no Arduino

Em acréscimo aos operadores *bitwise* já referidos, existem mais dois operadores, “>>”e “<<”, de acordo com a seguinte sintaxe:

variável **>>** número de bits Operador *shift right*

variável **<<** número de bits Operador *shift left*

Estes operadores realizam o deslocamento dos bits de uma dada variável (ou constante), pelo número de posições especificado, para a direita ou para a esquerda, respetivamente. Com estes operadores e com os outros já conhecidos, é possível realizar-se o que se designa por **máscaras**.



□ Operadores “>>”e “<<” no Arduino

As máscaras são úteis para se simular o valor lógico dos vários bits num dado barramento digital, tendo como efeito correspondente, a ligação de fios num aparato de *hardware*.

Com o operador “>>”, os n bits mais à direita desaparecem e entram n “1”s ou “0”s do lado esquerdo, dependendo do sinal algébrico do número deslocado.

Com o operador “<<”, os n bits mais à esquerda desaparecem e entram n “0”s do lado direito.

Exemplo: $S_1 = 1, S_0 = 0$. Com $(S1 << 1 | S0)$ obtém-se B10 como pretendido.



□ Operadores “>>”e “<<” no Arduino

Os operadores “>>”e “<<” também podem ser usados para realizar de forma muito rápida e computacionalmente eficiente, divisões e multiplicações por potências de 2, respetivamente. Por cada unidade de deslocamento, ter-se-á um decréscimo para metade, ou um acréscimo para o dobro, respetivamente.

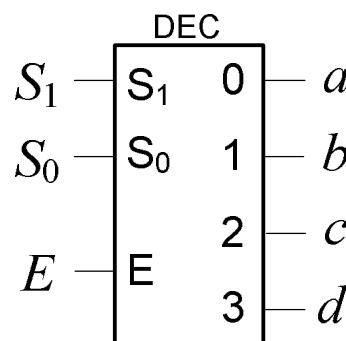
Exemplos:

```
int x = -16;      // B1111111111110000
int y = x >> 3; // B1111111111111110, decimal: -2
int w = 1000;    // B000000111101000
int z = w << 2; // B000011110100000, decimal: 4000
```



□ Decoder no Arduino

A implementação de *decoders* no Arduino pode ser feita segundo os mesmos princípios utilizados anteriormente para as outras funções lógicas. Num decoder que opera só com variáveis do tipo boolean (bits simples), a função de saída, simulada no Arduino, deve devolver um byte, sendo este uma composição dos vários bits de saída na posição certa. Possível implementação:



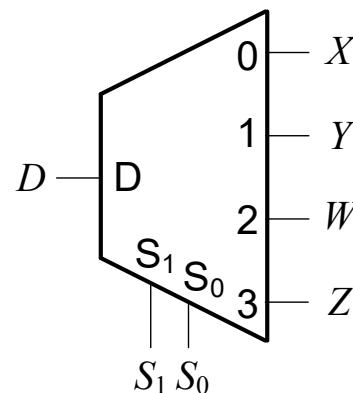
Decoder com dois bits de seleção

```
byte DEC2 (boolean S1, boolean S0, boolean E) {  
    return E << (S1 << 1 | S0);  
}
```



□ Demultiplexers no Arduino

A implementação de DMUXs no Arduino pode ser feita segundo os mesmos princípios utilizados para o *decoder*. A diferença reside na entrada de dados, a qual pode ocasionar, na saída selecionada, o valor “0” ou “1” consoante o valor desta entrada. Possível implementação (para o DMUX 1×4):



DMUX 1×4

```
byte DMUX_1x4 (boolean S1, boolean S0, boolean D) {  
    return D << (S1 << 1 | S0);  
}
```

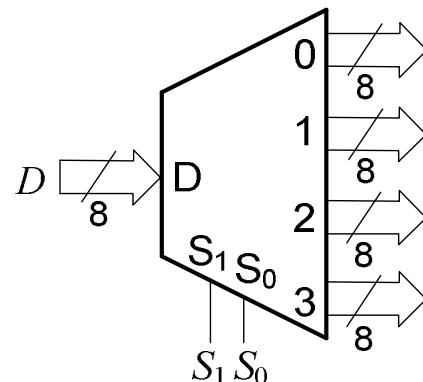
Qual é a conclusão a retirar, face ao *decoder* com *enable*?..



□ Demultiplexers no Arduino

Para o caso de um DMUX 1×4 que opere com variáveis do tipo byte, a função que o simula deverá articular-se com um *array* global que guarde os resultados das saídas do dispositivo. Uma implementação possível poderá ser:

DMUX 1×4



```
#define DIM_DMUX 4

byte DMUX_1x4_out[DIM_DMUX];

void DMUX_1x4 (boolean S1, boolean S0, byte D) {
    for (byte i = 0; i < DIM_DMUX; i++)
        DMUX_1x4_out[i] = 0;
    DMUX_1x4_out[S1 << 1 | S0] = D;
}
```



Circuitos sequenciais



□ Introdução

- Nos circuitos combinatórios, o valor lógico presente nas saídas é unicamente dependente do valor lógico presente, nesse momento, nas entradas.
- Nos circuitos sequenciais, o valor lógico presente nas **saídas** depende do valor das **entradas** e do **valor memorizado**. O valor memorizado depende dos valores das entradas a que o circuito esteve sujeito anteriormente. Existe, assim, a necessidade de que o circuito disponha de “memória” para se “recordar” dos valores que resultaram das entradas anteriores.



□ Tipos de células de memória

- *Latch*

Um *latch* é um dispositivo sequencial que verifica as suas entradas continuamente, e modifica as suas saídas em conformidade, em qualquer instante, independentemente de algum sinal de sincronismo

- *Flip-flop*

Um *flip-flop* é um dispositivo sequencial que verifica as suas entradas, e modifica as suas saídas, somente em instantes determinados por um sinal de sincronismo (*clock - CLK*).



□ Definição do *latch S-R*

Para contextualizar, a primeira célula de memória a abordar é o *latch S-R* (R : *reset* – colocar a “0”, S : *set* – colocar a “1”). A sua definição formal é:

- Sensível a dois sinais de entrada: S (*set*) para impor o estado 1, R (*reset*) para impor o estado “0”;
- Tem duas saídas: Q , cujo valor define o estado do *latch*, e \bar{Q} (complemento de Q);
- Mantém o estado anterior enquanto $S = 0$ e $R = 0$.
- Toma o valor 0, quando $S = 0$ e $R = 1$;
- Toma o valor 1, quando $S = 1$ e $R = 0$;
- A situação $S = 1$ e $R = 1$ não se utiliza.



□ Síntese do *latch S-R* a partir da definição

Tabela de verdade do *latch S-R*, de acordo com a definição anterior:

S	R	Q^*		Q
0	0	0		0
0	0	1		1
0	1	0		0
0	1	1		0
1	0	0		1
1	0	1		1
1	1	0		x
1	1	1		x

Q^* - Estado presente

Q - Estado seguinte

Q	R			
0	0	0	x	1
1	1	0	x	1

Q^* | \overline{S}

Se os *don't care* forem encarados como “1”, Q fica:

$$Q = S + \overline{R}.Q^* = \overline{S} + \overline{\overline{R}}.\overline{Q^*} = \overline{\overline{S}}.\overline{\overline{R}}.Q^*$$

Se os *don't care* forem encarados como “0”, Q fica:

$$Q = S.\overline{R} + \overline{R}.Q^* = \overline{S}.\overline{\overline{R}} + \overline{\overline{R}}.\overline{Q^*} = \overline{\overline{R}}(S + Q^*) = R + S + Q^*$$

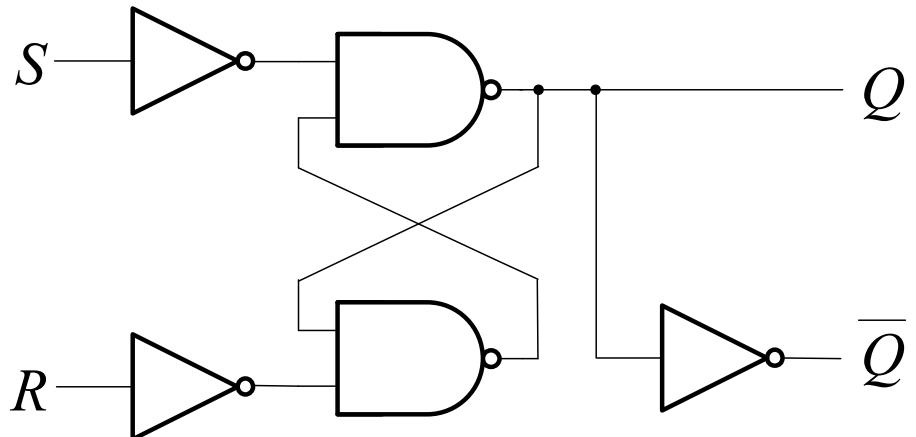


□ Implementação do *latch S-R* com portas NAND

A função $Q = \overline{\bar{S} \cdot \bar{R} \cdot Q^*}$

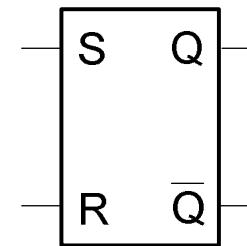
corresponde ao esquema:

(implementação com portas NAND)



A situação $R = 1$ e $S = 1$ não deve ser considerada porque, no caso concreto do presente circuito, Q ficará a 1, significando que a entrada *set* é prioritária sobre a *reset* – *set overrides reset*.

Símbolo lógico:

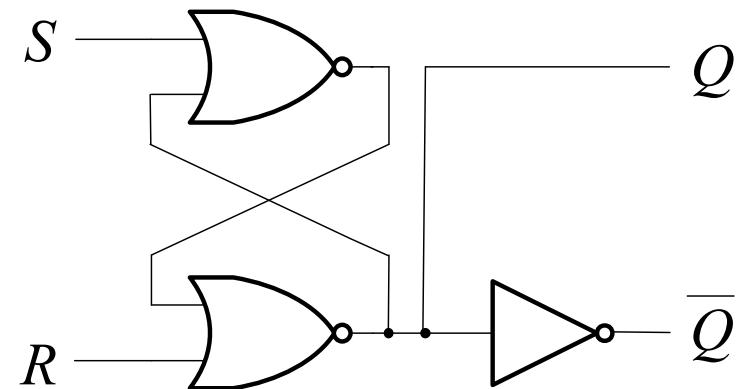


☐ Implementação do *latch S-R* com portas NOR

A função $Q = \overline{R} + \overline{S} + \overline{Q}^*$

corresponde ao esquema:

(implementação com portas NOR)



Nesta versão, tudo se passa como na versão realizada com portas NAND, exceto que, no caso em que $S = R = 1$, Q toma o valor 0, ou seja o *reset* é prioritário sobre o *set – reset overrides set*.

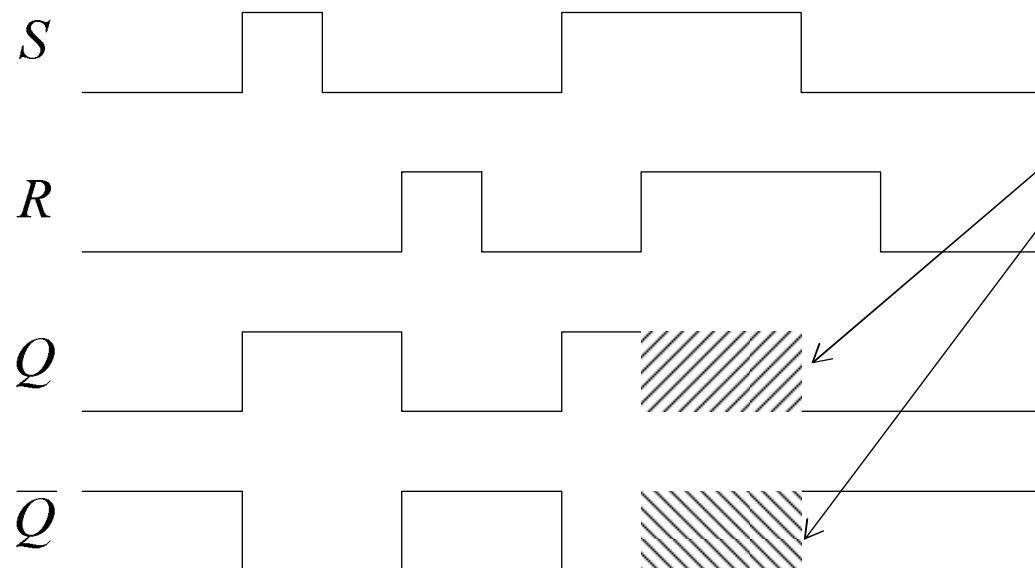
Na verdade, a situação $S = R = 1$ não conduz a um valor indeterminado, mas a um valor bem definido, que é função da implementação empregue.



□ Funcionamento do *latch S-R*

Comercialmente, poderá desconhecer-se como é que o *latch* é implementado, pelo que a combinação de entradas $S = R = 1$ é de excluir.

Diagrama temporal das saídas, função das entradas S e R :



Indeterminado, função da implementação com portas NAND ou NOR:

NAND $\rightarrow Q = 1$ e $\bar{Q} = 0$

NOR $\rightarrow Q = 0$ e $\bar{Q} = 1$



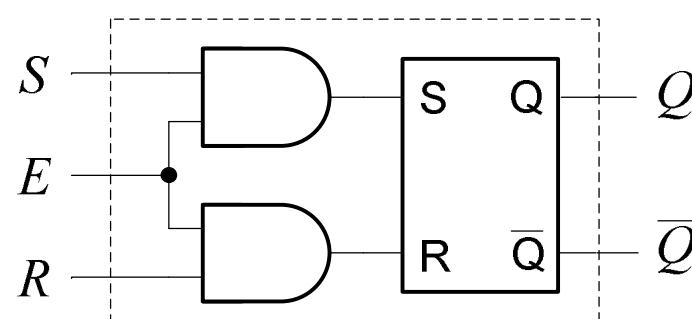
☐ Latch S-R-E (S-R com *enable*)

Definição:

- Enquanto $E = 0$, é insensível às entradas S e R , mantendo-se no estado anterior;
- Tem o comportamento do *latch S-R* quando $E = 1$.

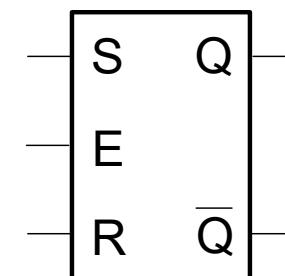
Tabela de verdade do *latch S-R-E*.

E	S	R	Q
0	-	-	Q^*
1	0	0	Q^*
1	0	1	0
1	1	0	1
1	1	1	-



Síntese a partir de um *S-R*.

Símbolo lógico:



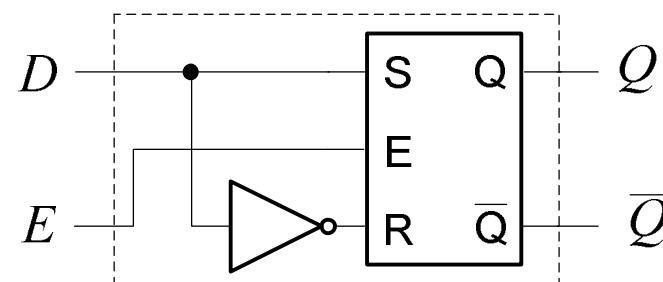


□ **Latch D**

Definição:

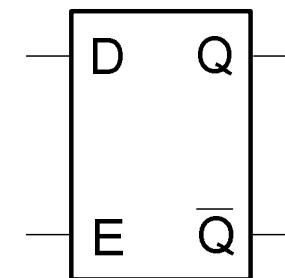
- Enquanto $E = 0$, mantém de memória o último valor assumido pela entrada D antes da transição descendente de E ;
- Mantém transparência da entrada D para a saída Q enquanto $E = 1$.

E	D	Q
0	0	Q^*
0	1	Q^*
1	0	0
1	1	1



Síntese a partir de um $S-R-E$.

Símbolo lógico:

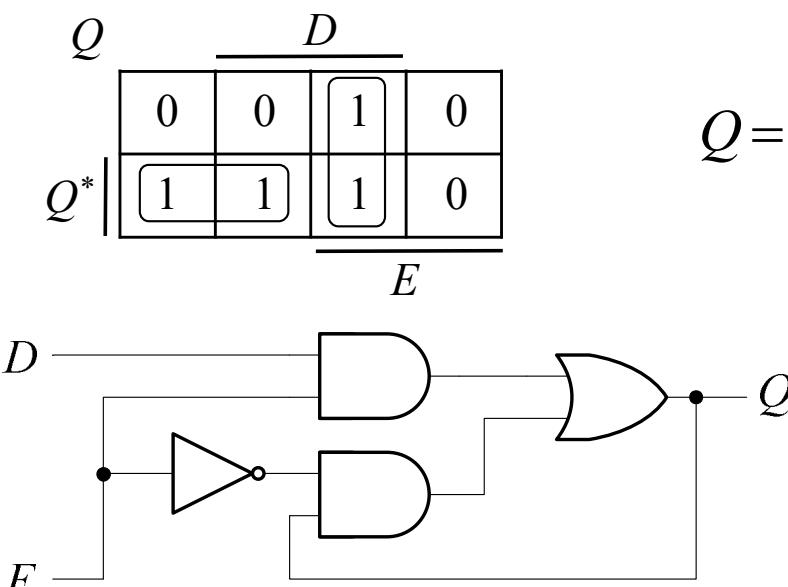




□ Síntese direta do *latch D*

A partir da tabela de verdade anterior, é possível criar uma tabela de verdade só com valores lógicos de 0 e 1 para Q , sintetizando-se o circuito lógico mais simples.

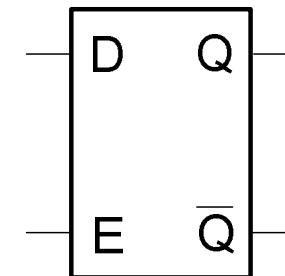
Q^*	E	D	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1



Síntese a partir da definição

$$Q = D \cdot E + Q^* \cdot \bar{E}$$

Símbolo lógico:

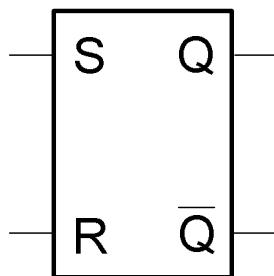




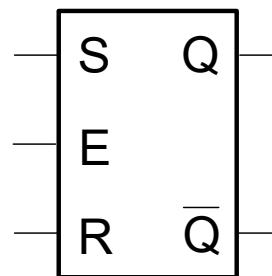
□ Resumo das células de memória do tipo “*latch*”

Os *latch* vistos até aqui, respondem de imediato aos estímulos que forem colocados nas suas entradas. Tal tipo de dispositivo designa-se por assíncrono.

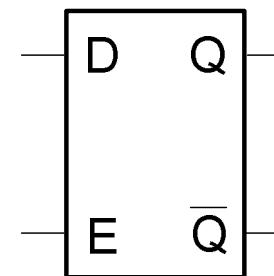
Símbolos lógico dos *latch* assíncronos já abordados:



Latch S-R



Latch S-E-R



Latch D



□ Conceito de *edge-triggered*

Ao contrário dos *latch*, os quais fazem sentir de imediato na sua saída os estímulos das entradas, nalgumas circunstâncias, interessa que as células de memória seja recetivas só em determinados instantes chave.

Por exemplo:

No *latch D*, interessa que a célula de memória seja sensível ao valor lógico presente na entrada *D*, não durante todo o tempo em que a entrada *E* esteja a 1, mas exclusivamente na sua transição de 1 para 0 – transição descendente de *E*. Dispositivos com este comportamento denominam-se “*flip-flops edge-triggered*”, também se denominando por síncronos.



□ Flip-flop “D edge-triggered”

Definição:

- Transfere a entrada D para a saída Q no instante da transição ascendente de *clock* (CLK), mantendo-a memorizada até que ocorra outra transição ascendente de *clock*.

	CLK	D	Q
Tabela de verdade	\uparrow	0	0
do flip-flop D	\uparrow	1	1
<i>edge triggered:</i>	0	-	Q^*
	1	-	Q^*
	\downarrow	-	Q^*

Tabela de transição de estados:

Q^*	Q	D
0	0	0
0	1	1
1	0	0
1	1	1



□ Flip-flop “D edge-triggered”

Símbolos lógicos possíveis dos *flip-flops D edge-triggered*:

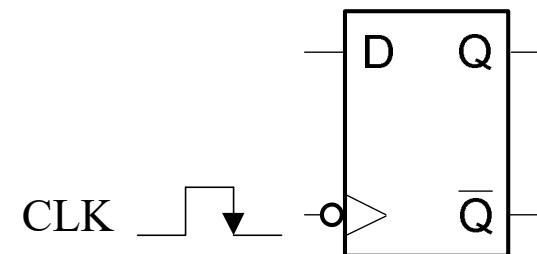
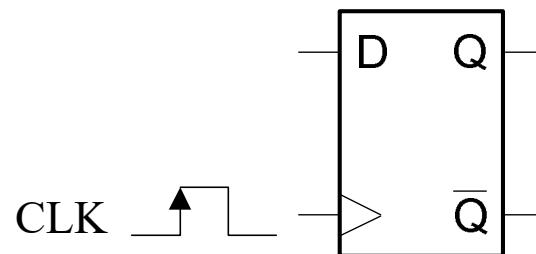
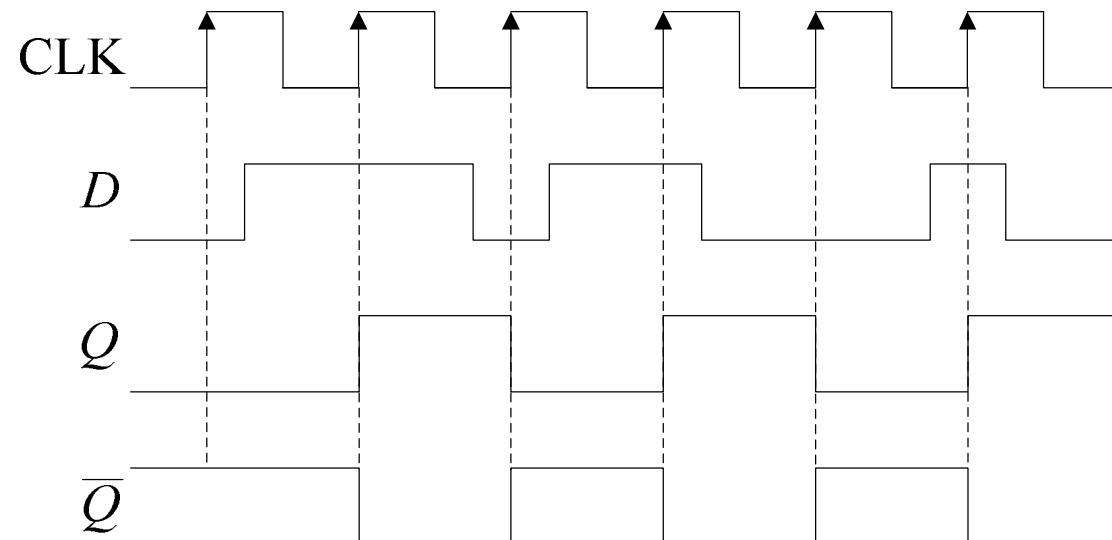


Diagrama temporal
de formas de onda:





Flip-flop “J-K edge-triggered”

Definição:

- Na transição ascendente de CLK, toma em consideração os valores presentes nas entradas J e K , de tal modo que, se $Q = 1$, só $K = 1$ poderá levá-lo para 0;
- Se estiver no estado $Q = 0$, só $J = 1$ poderá levá-lo para 1. Quando ambas as entradas J e K estiverem a 1, inverte o estado da saída quando acontece uma transição ascendente de CLK.



☐ Tabela de verdade do *flip-flop “J-K edge-triggered”*

Tabela de verdade do *flip-flop J-K edge-triggered*, a partir da definição:

Tabela de verdade do *flip-flop J-K edge triggered*:

CLK	J	K	Q
↑	0	0	Q^*
↑	0	1	0
↑	1	0	1
↑	1	1	\overline{Q}^*
0	-	-	Q^*
1	-	-	Q^*
↓	-	-	Q^*



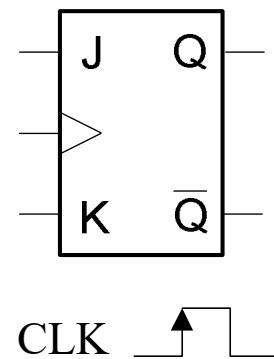
□ Símbolos do *flip-flop “J-K edge-triggered”*

A tabela de transição de estados é um auxiliar muito importante para realizar o projeto de circuitos lógicos envolvendo *flip-flops*.

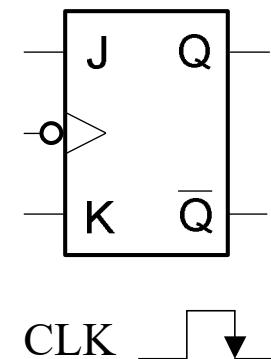
Tabela de transição
de estados:

Q^*	Q	J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

Símbolos lógicos possíveis para
os *flip-flops J-K edge-triggered*:



CLK

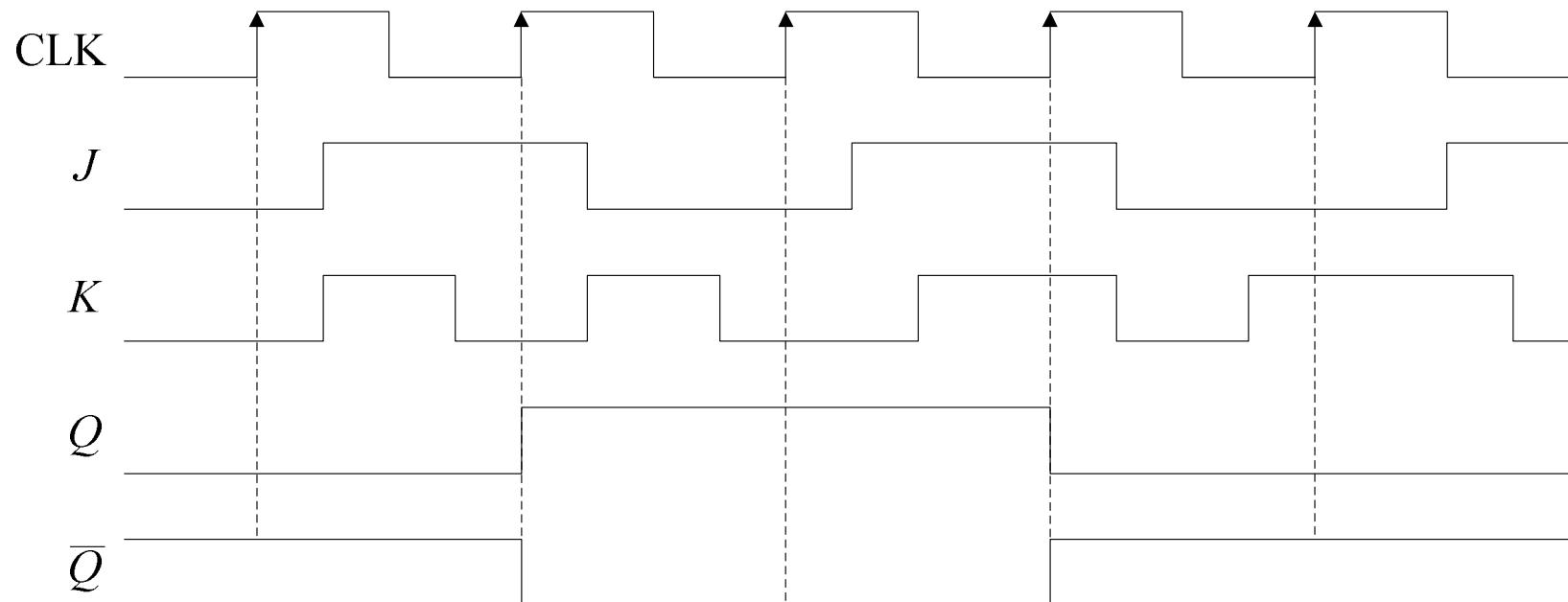
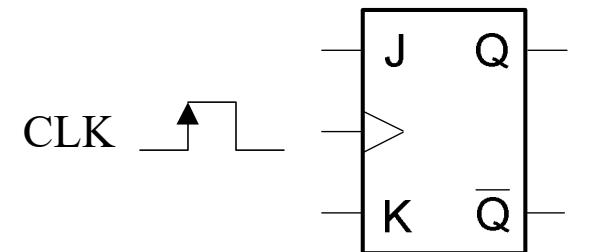


CLK



□ Exemplo de funcionamento do flip-flop “J-K edge-triggered”

Diagrama temporal de formas de onda:





☐ Flip-flop “T edge-triggered”

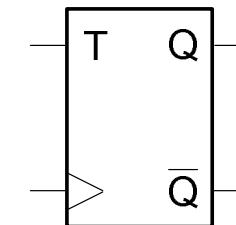
Definição:

- Sempre que $T = 1$, inverte o estado a cada transição ascendente de CLK;
- Com $T = 0$, permanece no estado anterior, estando insensível ao *clock*.

Tabela de verdade do *flip-flop T edge triggered*:

CLK	T	Q
\uparrow	0	Q^*
\uparrow	1	\bar{Q}^*
0	-	Q^*
1	-	\bar{Q}^*
\downarrow	-	Q^*

Símbolo lógico:



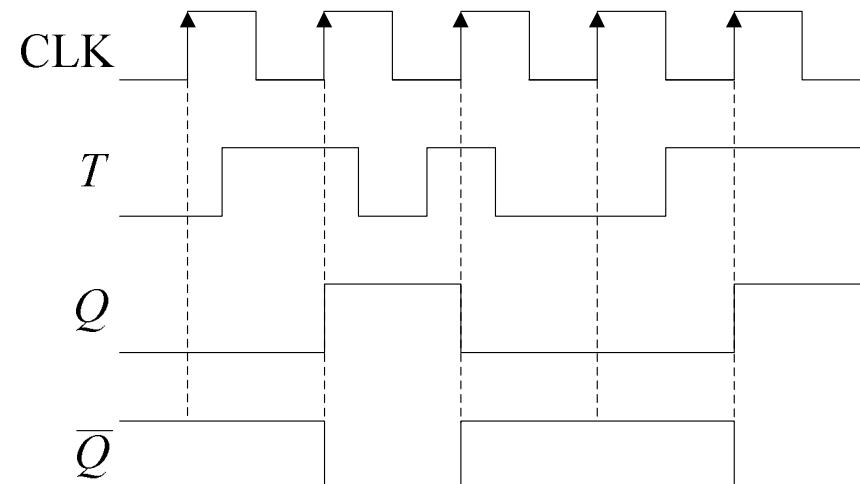


□ Exemplo de funcionamento do flip-flop “T edge-triggered”

Tabela de transição
de estados:

Q^*	Q	T
0	0	0
0	1	1
1	0	1
1	1	0

Diagrama temporal de
formas de onda:

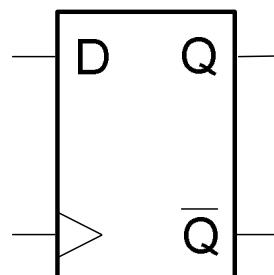




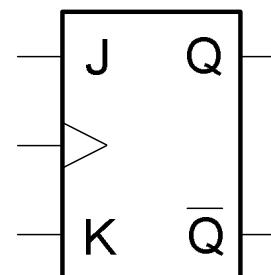
□ Resumo dos flip-flops tipo “edge-triggered”

Símbolos lógico dos flip-flops edge-triggered (síncronos) já abordados:

Flip-flop D



Flip-flop J-K



Flip-flop T

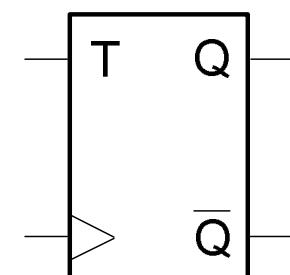


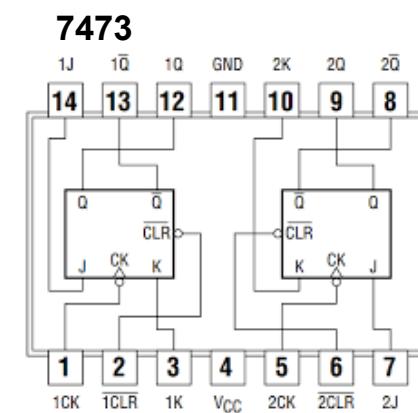
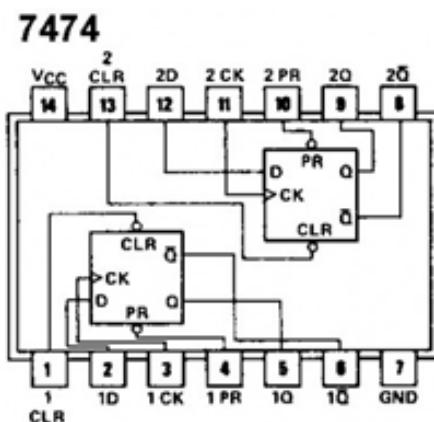
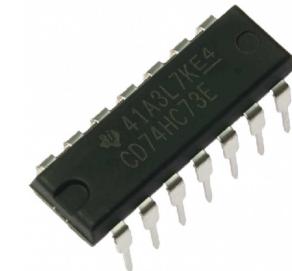
Tabela de transição de estados
dos flip-flops D, J-K e T:

Q^*	Q	D	J	K	T
0	0	0	0	-	0
0	1	1	1	-	1
1	0	0	-	1	1
1	1	1	-	0	0

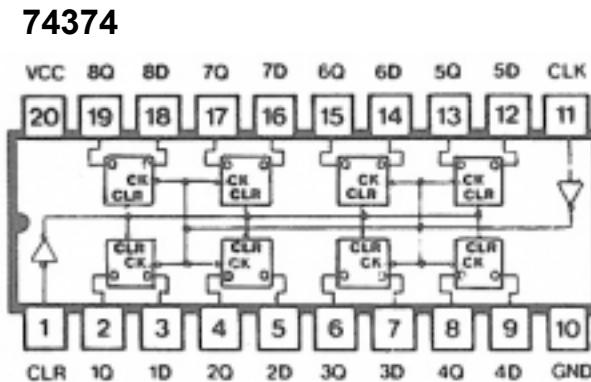


☐ Circuitos integrados

Tal como as portas lógicas já estudadas, os *flip-flops* existem comercialmente em circuito integrado.



Registro de 8 bits (1 byte)



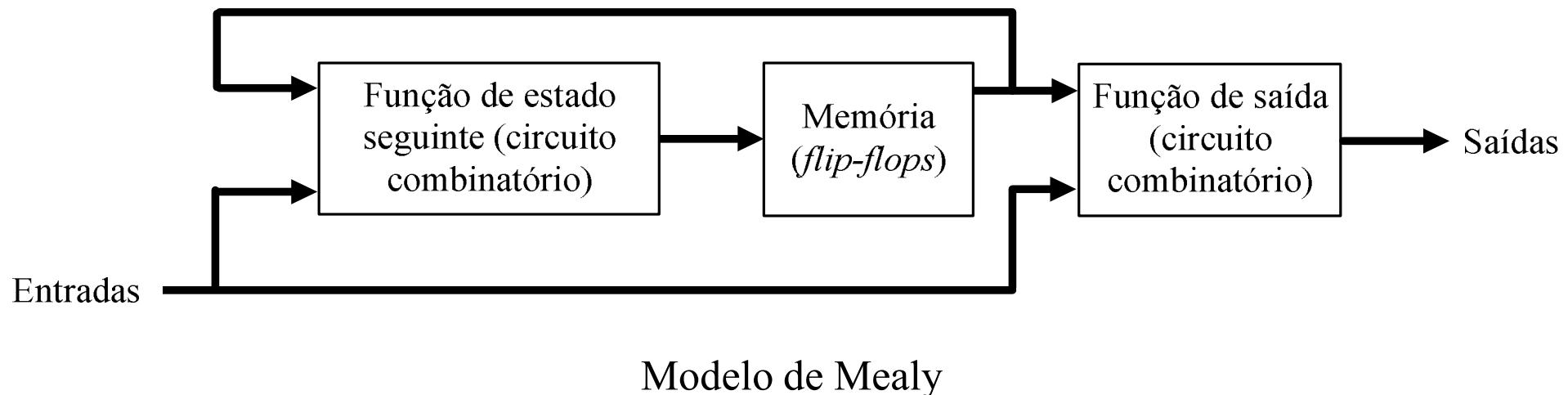
Alguns *flip-flops* dispõem de entradas assíncronas *preset* (PR) e *clear* (CLR), geralmente, *active low*.



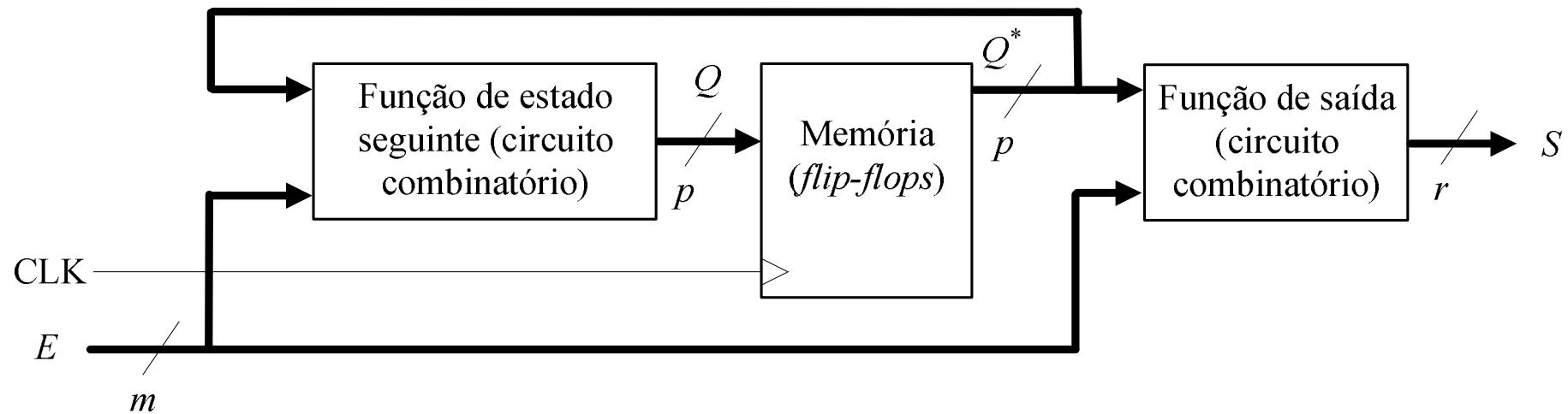
□ Projeto de circuitos sequenciais (máquinas de estados)

O projeto *hardware* de circuitos sequenciais é baseado em *flip-flops* e designa-se por máquinas de estados. Os modelos existentes são o modelo de Moore, de Mealy e modelo composto de Moore-Mealy.

Dar-se-á mais ênfase ao **modelo de Mealy**, dado que este é o que representa, de uma forma mais genérica, as máquinas de estados.



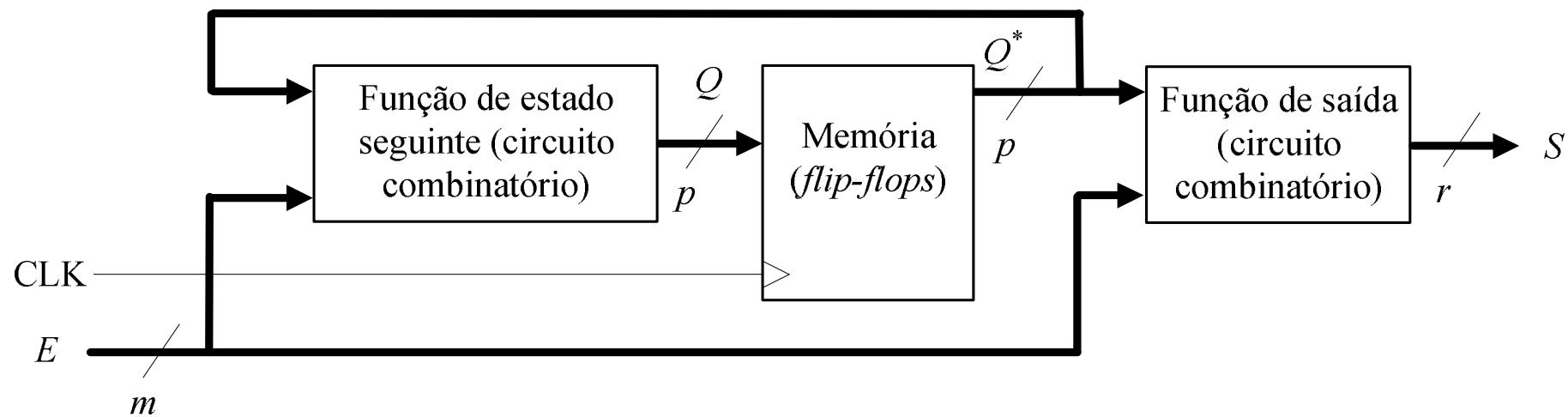
□ Projeto de circuitos sequenciais (máquinas de estados)



Os circuitos sequenciais prestam-se à adoção de métodos de projeto sistemáticos e englobam o conhecimento dos circuitos combinatórios.

A utilização do *ASM chart*, como passo intermédio de um projeto de circuito sequencial, separa o problema de formulação do algoritmo e da correspondente implementação.

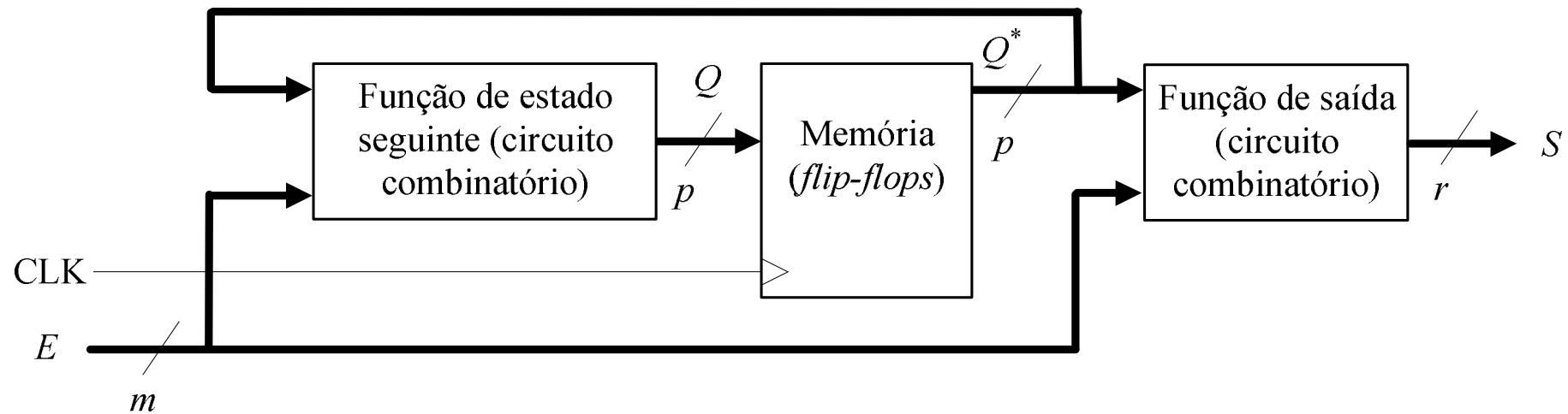
Projeto de circuitos sequenciais (máquinas de estados)



- O bloco de “Memória” é um conjunto de p *flip-flops edge-triggered*. Sendo estes do tipo D ou T , ter-se-ão p entradas, com $J-K$, serão $2 \times p$ entradas;
 - Q^* é o estado presente da memória (valores das saídas Q dos *flip-flops*);
 - Q é o conjunto de valores das entradas dos *flip-flops*, que vão dar como resultado o estado seguinte (para o qual o sistema vai evoluir na próxima transição de *clock*);

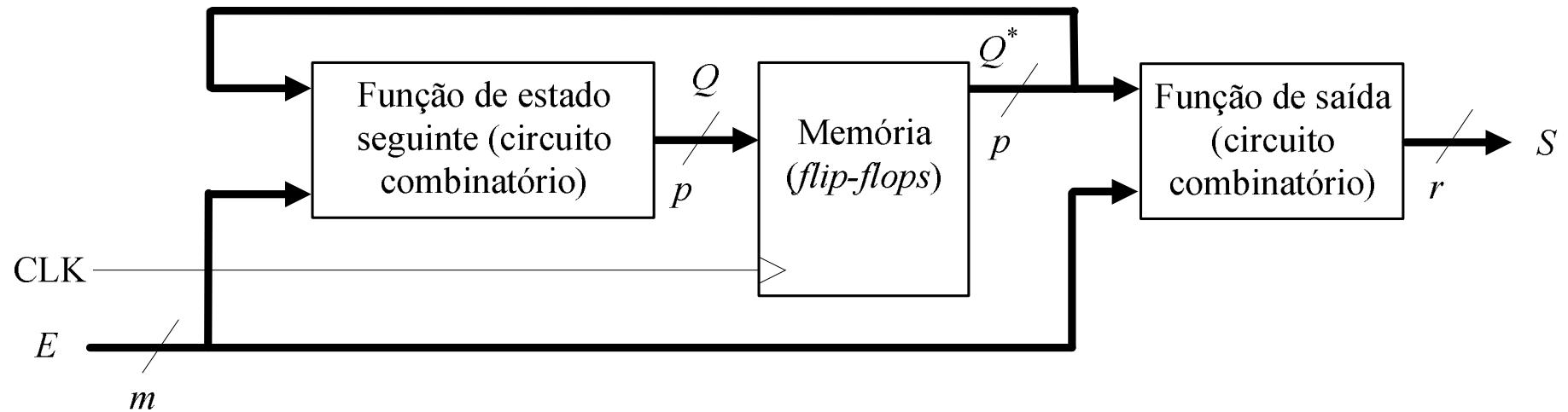


□ Projeto de circuitos sequenciais (máquinas de estados)



- $S = f(Q^*, E)$ é o conjunto de funções booleanas que relaciona as saídas (S) com o estado presente (Q^*) e as entradas (E);
- $Q = g(Q^*, E)$ é o conjunto de funções booleanas que relaciona o estado seguinte (Q) com o estado presente (Q^*) e as entradas (E);
- As p componentes de Q^* (estado presente da memória) chamam-se variáveis de estado (valores das saídas Q dos *flip-flops*);

□ Projeto de circuitos sequenciais (máquinas de estados)



- As funções f e g podem ser descritas por ASM ou por uma tabela de “Estado seguinte” e “Saída”, função de “Estado presente” e “Entrada”;

Q^*				E			Q				S				
Q_{p-1}^*	...	Q_1^*	Q_0^*	E_{m-1}	...	E_1	E_0	Q_{p-1}	...	Q_1	Q_0	S_{r-1}	...	S_1	S_0



□ Projeto de circuitos sequenciais (máquinas de estados)

Formas de representação de sistemas:

- Diagrama de blocos
- Diagrama de estados (ASM – *Algorithmic State Machine*)

Os sistemas são projetados para cumprir objetivos, realizando um algoritmo.

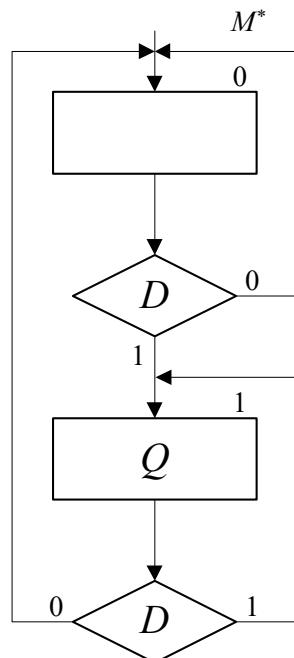
Algoritmo

- Processo suscetível de ser mecanizado (sistematizado), ou seja, implementado por dispositivos digitais;
- Propriedades: finito, inteligível (não existirem ambiguidades), exequível, caracterizável externamente (relação entre saídas e entradas);
- Formas de representação mais utilizadas: fluxograma, ASM *chart* (para representação de máquinas de estados).

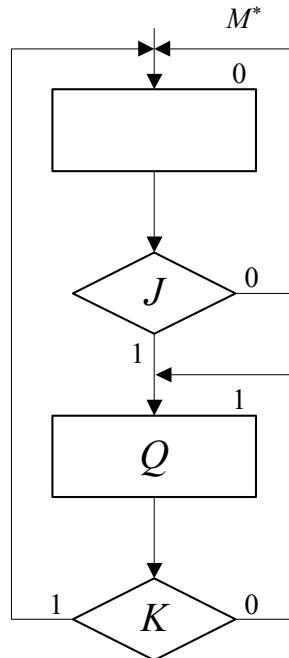
□ Diagramas ASM dos *flip-flops*

Trata-se sempre de *flip-flops edge-triggered*. O teste do valor lógico das entradas, que condicionará o próximo estado, ocorre sempre na transição ascendente de *clock*. A variável de estado é M^* , para não se confundir com a saída Q .

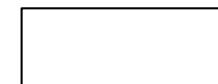
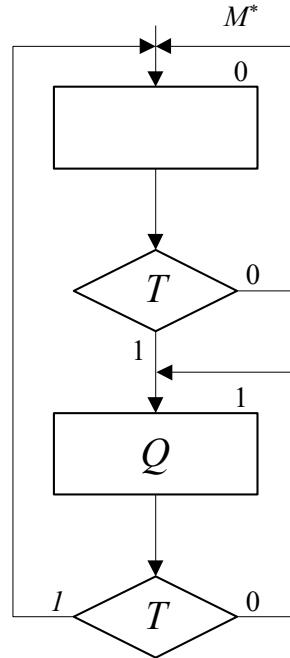
Flip-flop D



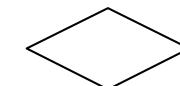
Flip-flop J-K



Flip-flop T



Estado



Teste sobre uma variável de entrada
(no momento da transição do *clock*)



□ Contadores

Os contadores são dispositivos destinados a realizar contagens em sequência, possuindo um registo (conjunto de *flip-flops*), sobre o qual a contagem é acumulada.

Caracterização dos contadores

Estrutura
 { Síncronos
 Assíncronos

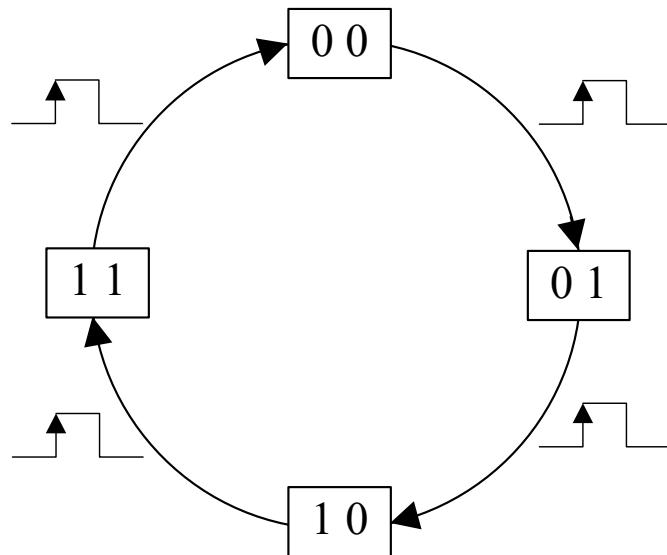
Módulo de contagem
 { Potência de 2
 Diferente de potência de 2

Sequência de contagem
 { Binária natural
 { Crescente
 Decrescente
 Crescente ou Decrescente
 Outras sequências (Ex.: Código Gray)

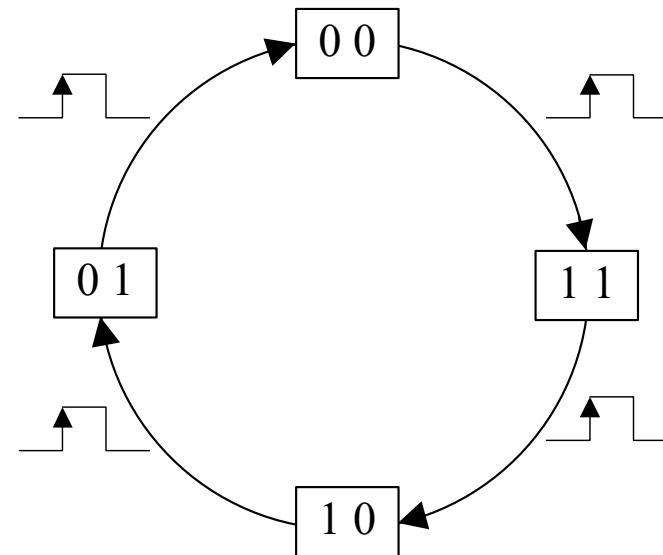


□ Contadores

Evolução da sequência de contagem num contador síncrono, módulo 4, com sequências de contagem crescente ou decrescente, respetivamente.



Crescente



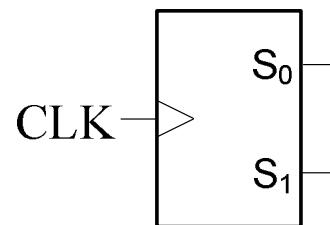
Decrescente



□ Projeto estruturado de um contador módulo 3, crescente

Este contador terá três estados de contagem, necessitando portanto de dois *flip-flops* para fornecer os bits de saída.

Modelo “caixa preta”
do contador módulo 3



S_1	S_0
0	0
0	1
1	0

0	0
...	

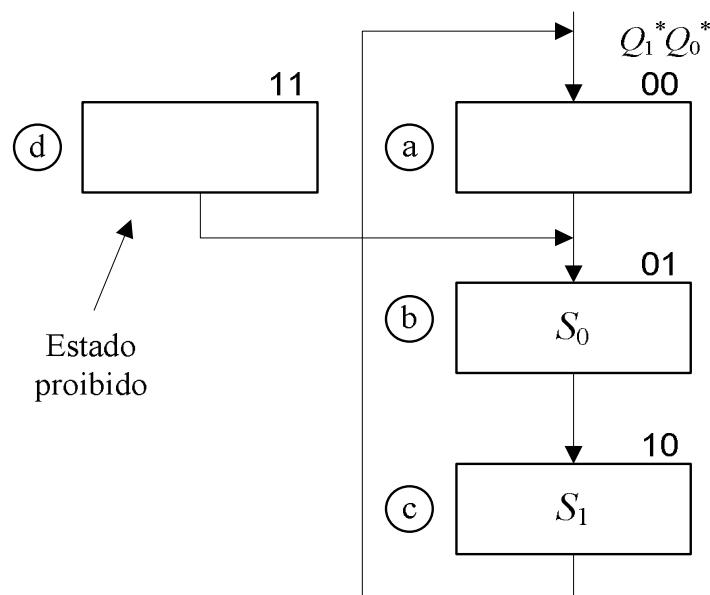
Tabela de sequência
de estados e saídas

Q^*		Q		S	
Q_1^*	Q_0^*	Q_1	Q_0	S_1	S_0
0	0	0	1	0	0
0	1	1	0	0	1
1	0	0	0	1	0
1	1	0	1	0	0

A saída tomará três estados: 00, 01 e 10. Apesar de 11 ser possível, não é utilizado, arbitrando-se que transita para o estado 01. Se 11, eventualmente, surgir quando a alimentação é ligada, o utilizador não notará diferença na sequência presente na saída.

□ Projeto estruturado de um contador módulo 3, crescente

ASM do contador



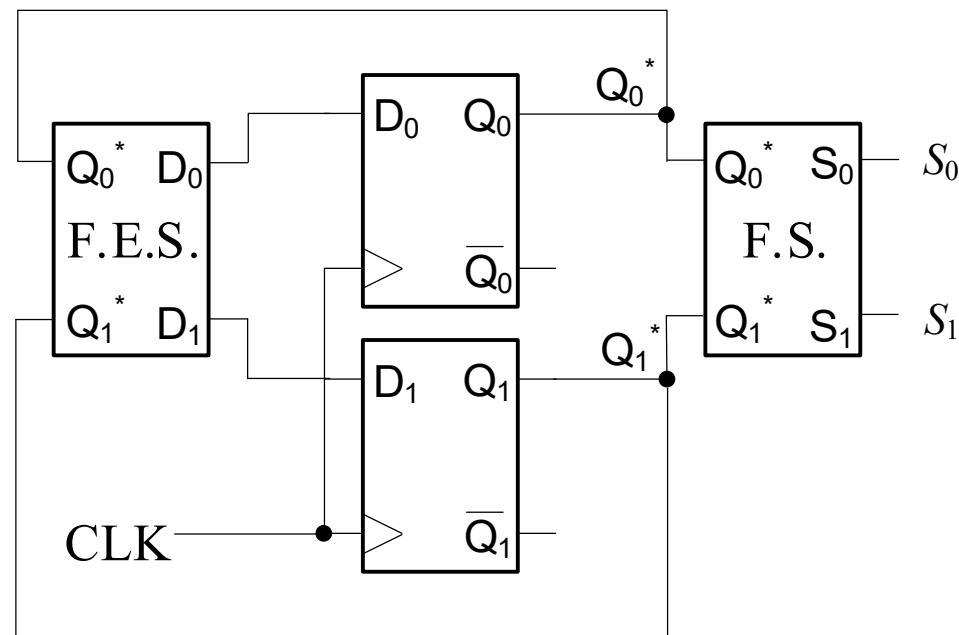
- Os estados são representados por retângulos, atribuindo a cada um uma mnemónica que fica do lado esquerdo;
- A cada estado, é atribuído um código binário (*state assignment*), em que cada bit traduz o estado dos *flip-flops* do registo, sendo indicado por cima do retângulo, no lado direito;
- Dentro do retângulo, indica-se o nome das saídas ativas para o respetivo estado.

As saídas precisam de lógica de descodificação, para se obterem os valores de S , em função de Q^* , tendo-se saídas função de estado.



□ Projeto estruturado de um contador módulo 3, crescente

Diagrama de blocos do contador, no qual se têm as funções de estado seguinte (FES) para cada um dos *flip-flops* e a função de saída (FS).



A partir do estado atual, a função de estado seguinte (FES) é um circuito combinatório que determina qual será o estado seguinte, de cada *flip-flop*, e FS é a função de saída.



□ Projeto estruturado de um contador módulo 3, crescente

Síntese com *flip-flops D*

Funções de estado seguinte,
na síntese com *flip-flops D*:

Atribuição de estados
(*state assignment*)

$\overline{Q_0^*}$	
a	b
c	d

D_1	$\overline{Q_0^*}$	
0	1	
0	0	

$$D_1 = \overline{Q_1^*} \cdot Q_0^*$$

D_0	$\overline{Q_0^*}$	
1	0	
0	1	

$$D_0 = \overline{Q_1^*} \oplus Q_0^*$$

Q_1^*	Q_0^*	D_1	D_0
0	0	0	1
0	1	1	0
1	0	0	0
1	1	0	1

Tabela de transição de
estados do *flip-flop D*

Q^*	Q	D
0	0	0
0	1	1
1	0	0
1	1	1

Por observação do ASM e da
tabela de transição de estados dos
flip-flops, preenchem-se os
mapas de Karnaugh de cada
entrada dos *flip-flops*, para a
transição de cada um dos estados.

□ Projeto estruturado de um contador módulo 3, crescente

Síntese com *flip-flops D*

Funções de saída, na síntese com *flip-flops D*:

(É a mesma para os outros tipos de *flip-flops*)

Funções de saída

Q_1^*	Q_0^*	S_1	S_0
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	0

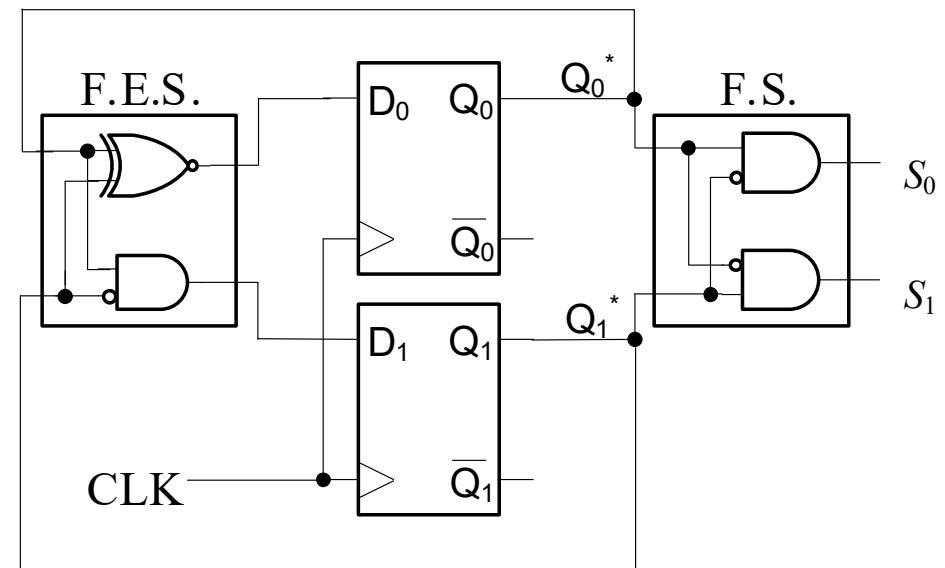
Q_0^*		Q_1^*
0	0	
1	0	

$S_1 = Q_1^* \cdot \overline{Q_0^*}$

Q_0^*		Q_1^*
0	1	
0	0	

$S_0 = \overline{Q_1^*} \cdot Q_0^*$

Circuito lógico final:



FES com duas saídas: D_0 e D_1



□ Projeto estruturado de um contador módulo 3, crescente

Síntese com *flip-flops J-K*

Funções de estado seguinte,
na síntese com *flip-flops J-K*:

Q_1^*	Q_0^*	J_1	K_1	J_0	K_0
0	0	0	-	1	-
0	1	1	-	-	1
1	0	-	1	0	-
1	1	-	1	-	0

J_1	$\overline{Q_0^*}$		K_1	$\overline{Q_0^*}$	
0	0	1	-	-	-
-	-	-	1	1	1

$$J_1 = Q_0^*$$

J_0	$\overline{Q_0^*}$		K_0	$\overline{Q_0^*}$	
1	1	-	-	1	-
0	0	-	-	0	-

$$J_0 = \overline{Q_1^*}$$

$$K_0 = \overline{Q_1^*}$$

Tabela de transição
de estados dos *flip-*
flops J-K

Q^*	Q	J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

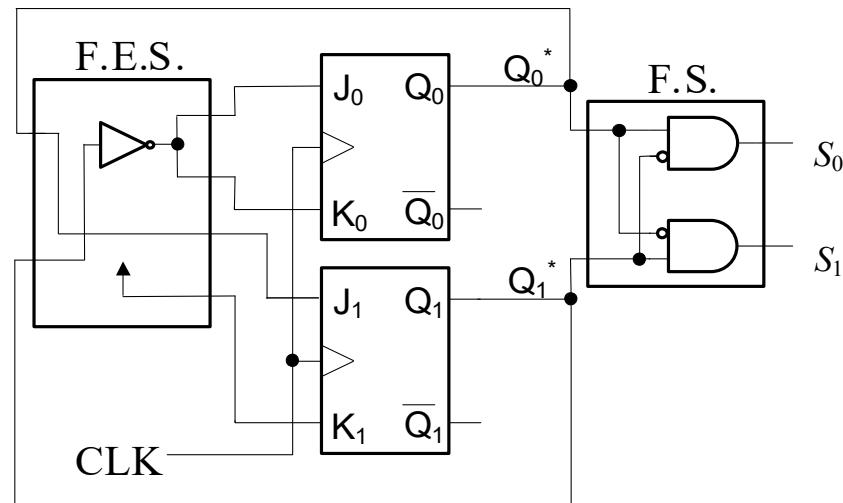


□ Projeto estruturado de um contador módulo 3, crescente

Síntese com *flip-flops J-K*

Circuito lógico final:

FES com quatro saídas:
 J_0, K_0, J_1 e K_1



Na realidade, nesta situação, o bloco combinatório de FES poderia não usar portas lógicas, dado que nos *flip-flops* se dispõe do complemento da sua saída.

A função de estado seguinte é a mesma já determinada, dado que esta não depende do tipo de *flip-flop* no qual o sistema é baseado.



□ Projeto estruturado de um contador módulo 3, crescente

Síntese com *flip-flops T*

Funções de estado seguinte,
na síntese com *flip-flops T*:

Atribuição de estados
(*state assignment*)

$\overline{Q_0^*}$	
a	b
c	d

T_1

$\overline{Q_0^*}$	
0	1
1	1

$$T_1 = Q_1^* + Q_0^*$$

T_0

$\overline{Q_0^*}$	
1	1
0	0

$$T_0 = \overline{Q_1^*}$$

Q_1^*	Q_0^*	T_1	T_0
0	0	0	1
0	1	1	1
1	0	1	0
1	1	1	0

Tabela de transição de
estados do *flip-flop T*

Q^*	Q	T
0	0	0
0	1	1
1	0	1
1	1	0

Por observação do ASM e da
tabela de transição de estados dos
flip-flops, preenchem-se os
mapas de Karnaugh de cada
entrada dos *flip-flops*, para a
transição de cada um dos estados.



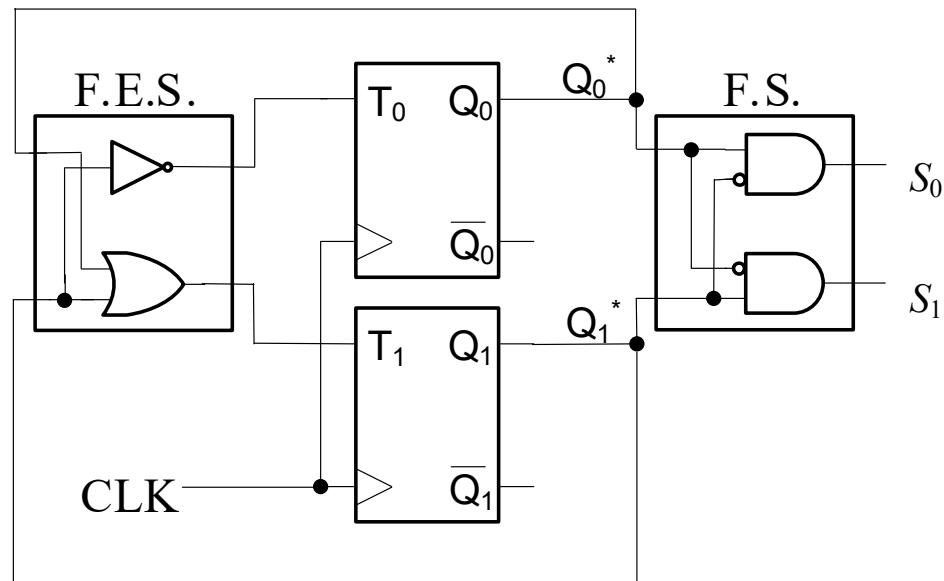
□ Projeto estruturado de um contador módulo 3, crescente

Síntese com *flip-flops T*

Círcuito lógico final:

FES com duas saídas:

T_0 e T_1



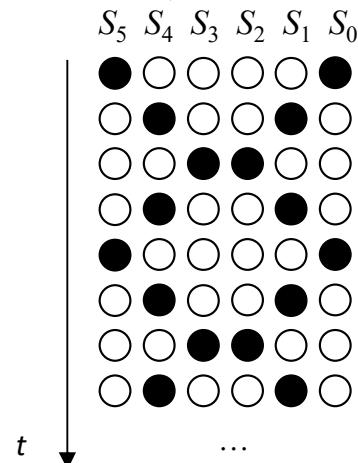
Também nesta situação, o inversor que liga de Q_1 a T_0 poderia ser omitido, utilizando-se diretamente a saída complementar de Q_1 .

A função de estado seguinte é a mesma já determinada, dado que esta não depende do tipo de *flip-flop* no qual o sistema é baseado.



□ Projeto de um circuito gerador de padrões sequenciais

É objetivo projetar uma máquina de estados que, ao ritmo dos impulsos de *clock*, acenda seis LEDs de acordo com o seguinte padrão:



Apesar de se terem seis saídas, o número de estados não é necessariamente $2^6 = 64$, dado que só existem três padrões diferentes que se repetem ciclicamente.

Contudo, também não se pode dizer que o número de estados seja igual a três, porque existem, nesse caso, transições de estado ambíguas.

Q^*	Q	S_5	S_4	S_3	S_2	S_1	S_0
a	b	1	0	0	0	0	1
b	c	0	1	0	0	1	0
c	b	0	0	1	1	0	0
b	a	0	1	0	0	1	0

Considerando apenas três estados, pode notar-se que existe ambiguidade na transição de **b** para outro estado: num caso, transita para **c** (2^a linha) e noutro, transita para **a** (4^a linha).



□ Projeto de um circuito gerador de padrões sequenciais

Como três estados implicam ambiguidade, significa que tem de se aumentar o número de estados. Com quatro estados, já não se têm ambiguidades:

Q^*	Q	S_5	S_4	S_3	S_2	S_1	S_0
a	b	1	0	0	0	0	1
b	c	0	1	0	0	1	0
c	d	0	0	1	1	0	0
d	a	0	1	0	0	1	0

Apesar de **b** e **d** terem a mesma configuração de saída, na verdade, são estados distintos, uma vez que transitam para um estado sucessor diferente. Como temos quatro estados, estes são codificados com dois bits.

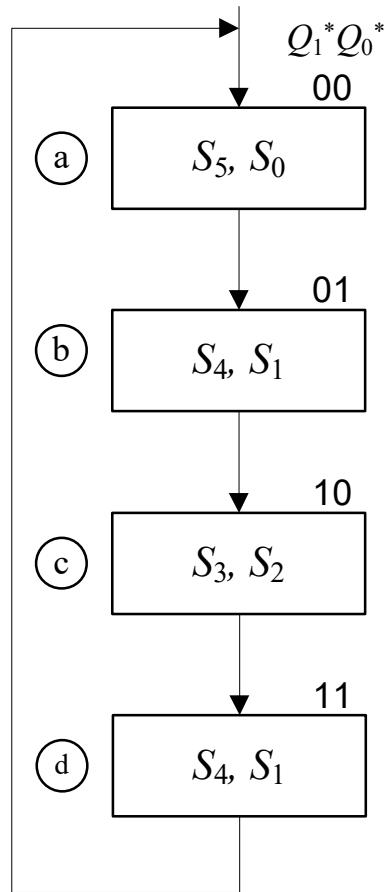
Q^*		Q		S					
Q_1^*	Q_0^*	Q_1	Q_0	S_5	S_4	S_3	S_2	S_1	S_0
0	0	0	1	1	0	0	0	0	1
0	1	1	0	0	1	0	0	1	0
1	0	1	1	0	0	1	1	0	0
1	1	0	0	0	1	0	0	1	0

Note-se que $S_5 = S_0$, $S_4 = S_1$ e $S_3 = S_2$, pelo que ter-se-á que descodificar cada um destes três padrões a partir de Q^* .



□ Projeto de um circuito gerador de padrões sequenciais

ASM chart



Mapa de state assignment

$\overline{Q_0}^*$	
a	b
c	d

$| Q_1^*$

Tabela de transição de estados do flip-flop D:

Q^*	Q	D
0	0	0
0	1	1
1	0	0
1	1	1

Síntese com flip-flops D

D_1	$\overline{Q_0}^*$
0	1
1	0

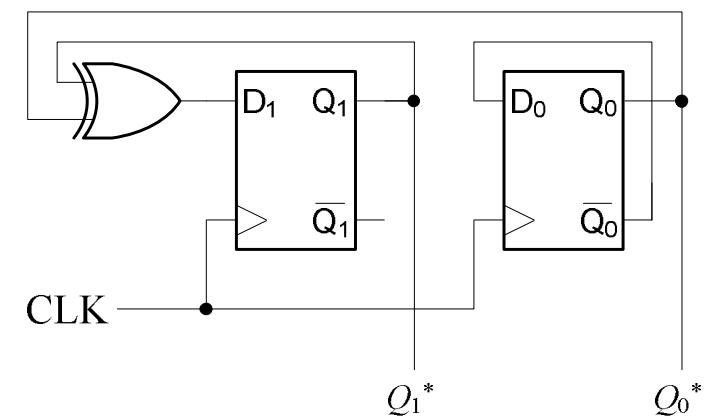
$| Q_1^*$

D_0	$\overline{Q_0}^*$
1	0
1	0

$| Q_1^*$

$$D_1 = Q_1^* \oplus Q_0^*$$

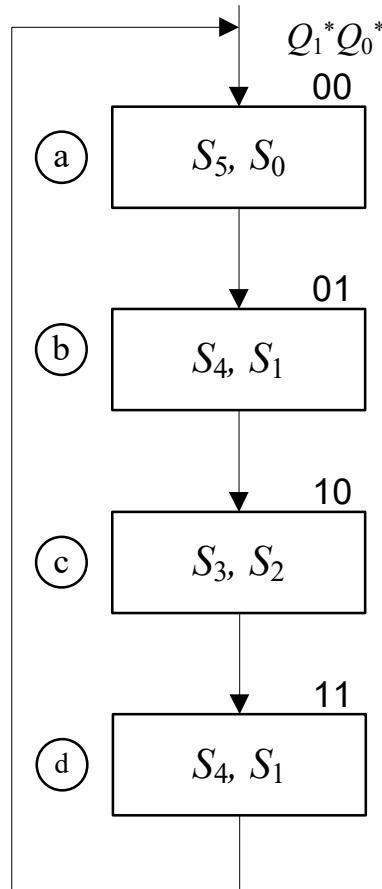
$$D_0 = \overline{Q_0}^*$$





□ Projeto de um circuito gerador de padrões sequenciais

ASM chart



Síntese com *flip-flops J-K*

J_1	$\underline{Q_0^*}$	
0	1	
-	-	$ Q_1^*$

$$J_1 = Q_0^*$$

K_1	$\underline{Q_0^*}$	
-	-	
0	1	$ Q_1^*$

$$K_1 = Q_0^*$$

J_0	$\underline{Q_0^*}$	
1	-	
1	-	$ Q_1^*$

$$J_0 = 1$$

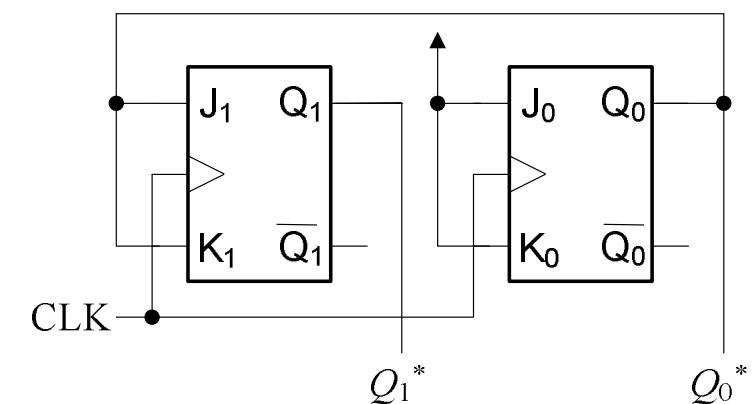
K_0	$\underline{Q_0^*}$	
-	1	
-	1	$ Q_1^*$

$$K_0 = 1$$

Mapa de state assignment

Q_0^*	Q	J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

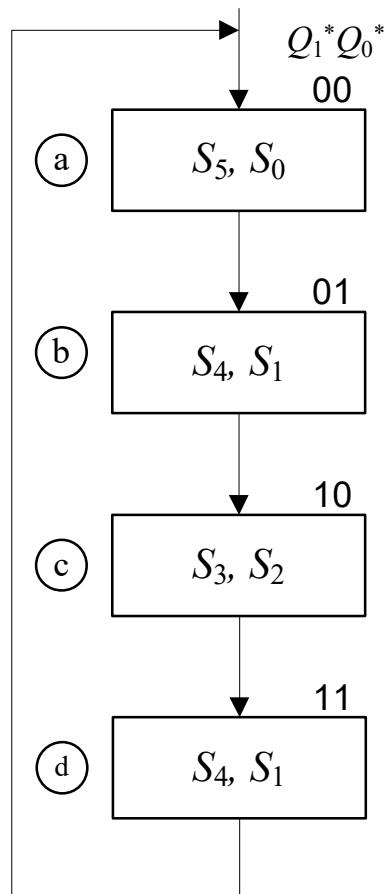
Q^*	Q	J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0





□ Projeto de um circuito gerador de padrões sequenciais

ASM chart



Mapa de state assignment

$\overline{Q_0}^*$	
a	b
c	d

$| Q_1^*$

Tabela de transição de estados do flip-flop T :

Q^*	Q	T
0	0	0
0	1	1
1	0	1
1	1	0

Síntese com flip-flops T

T_1	$\overline{Q_0}^*$
0	1
0	1

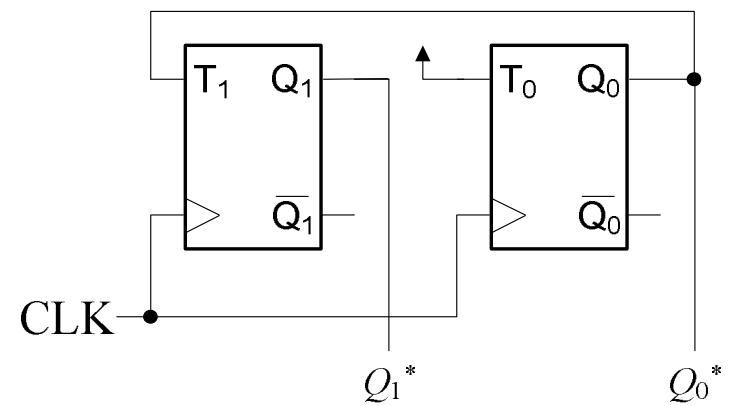
$| Q_1^*$

$T_1 = Q_0^*$

T_0	$\overline{Q_0}^*$
1	1
1	1

$| Q_1^*$

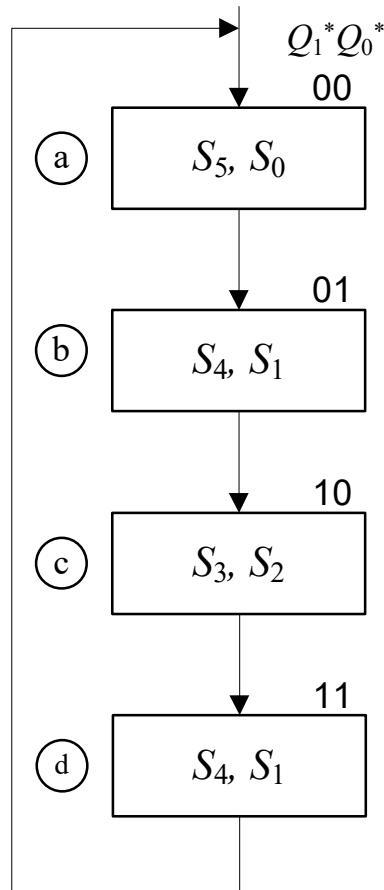
$T_0 = 1$





□ Projeto de um circuito gerador de padrões sequenciais

ASM chart

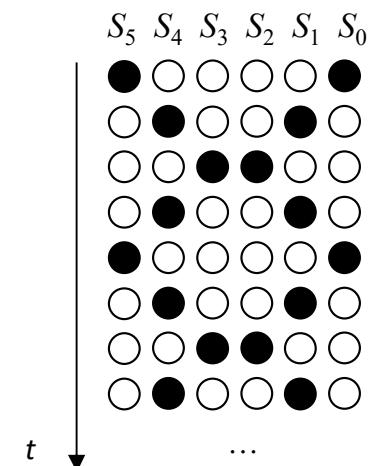
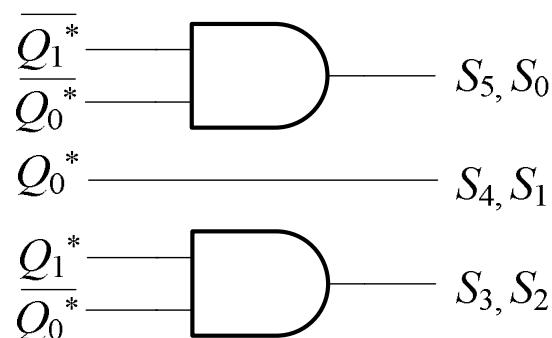


Descodificação das saídas, função das variáveis de estado:

$$S_5 = S_0 = \overline{Q}_1^* \overline{Q}_0^*$$

$$S_4 = S_1 = \overline{Q}_1^* \cdot Q_0^* + Q_1^* \cdot \overline{Q}_0^* = Q_0^* (\overline{Q}_1^* + Q_1^*) = Q_0^*$$

$$S_3 = S_2 = Q_1^* \overline{Q}_0^*$$



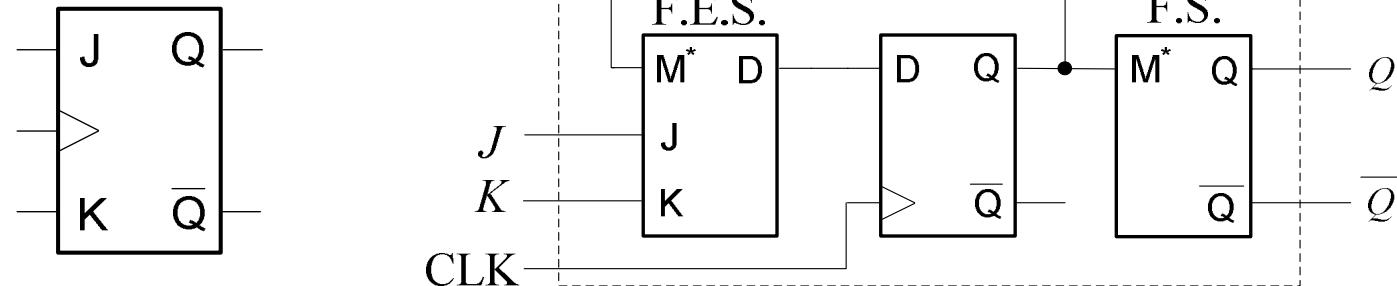


☐ Síntese do *flip-flop* J-K a partir do *flip-flop* D

É possível obter um dado tipo de *flip-flop* a partir de outro tipo, por exemplo, obter um *J-K* a partir de um *D*.

O processo de síntese baseia-se no ASM do *flip-flop* pretendido, tendo que determinar-se a FES e a FS do *flip-flop* em que se baseia a síntese.

Neste caso, tem-se uma nova situação, na qual a FES depende de variáveis externas (*J* e *K*), para além de depender da variável de estado.





□ Síntese do *flip-flop* J-K a partir do *flip-flop* D

É necessário ter em atenção o ASM do *flip-flop* final e a tabela de transição de estados do *flip-flop* na qual se baseia a síntese.

ASM do
flip-flop J-K

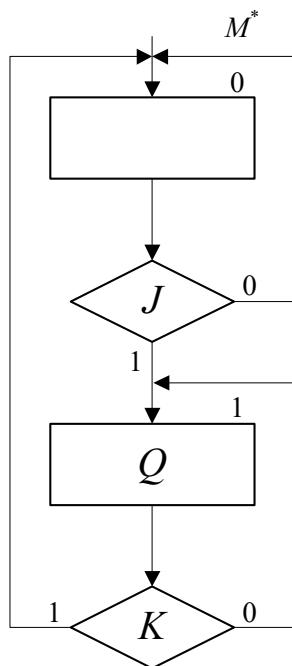


Tabela de transição de estados do *flip-flop* D:

Q^*	Q	D
0	0	0
0	1	1
1	0	0
1	1	1

M^* é a variável de estado (a 1 bit).

Funções de estado seguinte e de saída

M^*	J	K	D	Q
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	1
1	0	1	0	1
1	1	0	1	1
1	1	1	0	1

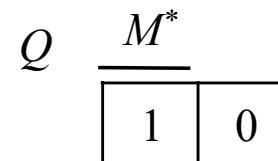
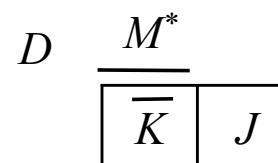


□ Síntese do *flip-flop* J-K a partir do *flip-flop* D

A tabela contendo as funções de estado seguinte e de saída pode ser condensada, fazendo-se a inserção das variáveis de entrada J e K .

M^*	J	K	D	Q
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	1
1	0	1	0	1
1	1	0	1	1
1	1	1	0	1

M^*	D	Q
0	J	0
1	\bar{K}	1



$$D = M^* \cdot \bar{K} + \bar{M}^* \cdot J$$

$$Q = M^*$$

O preenchimento do mapa de Karnaugh faz-se recorrendo à inserção das variáveis de entrada.

Como, no *flip-flop* D, existe identidade entre D e Q , escolhe-se o valor das variáveis que conduz ao valor de estado seguinte igual a 1.



□ Mapas de Karnaugh com variáveis inseridas

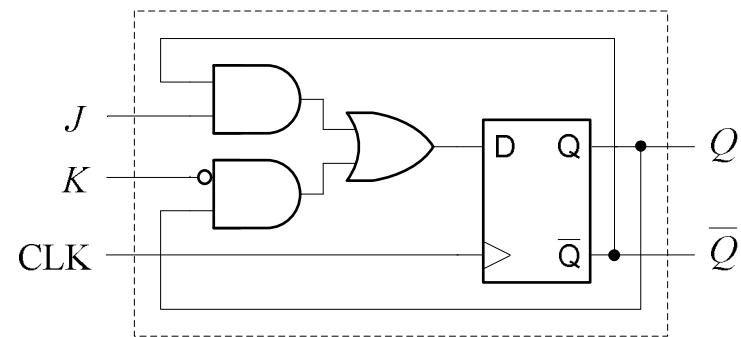
Extração de expressões simplificadas em mapas de Karnaugh com variáveis inseridas:

- Extrair os “1”s presentes no mapa, considerando como “0” as variáveis inseridas, e aproveitando os eventuais *don't care* para simplificação;
- Extrair as variáveis inseridas, considerando também os “1”s como *don't care*.

Síntese de um *flip-flop J-K*
a partir de um *flip-flop D*

$$D = M^* \cdot \overline{K} + \overline{M}^* \cdot J$$

$$Q = M^*$$



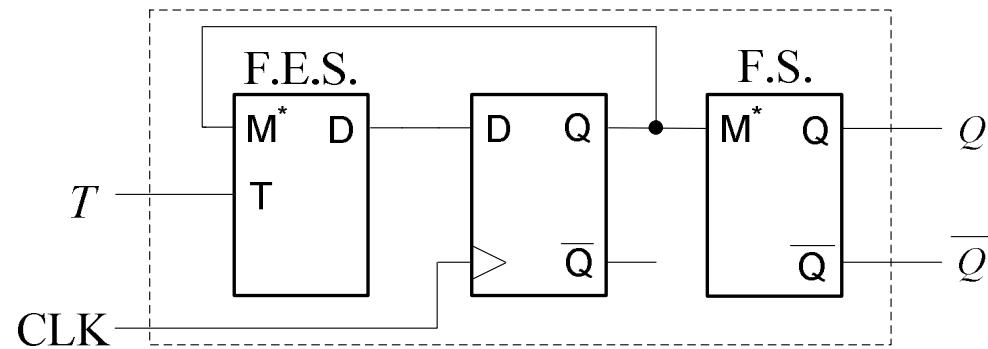
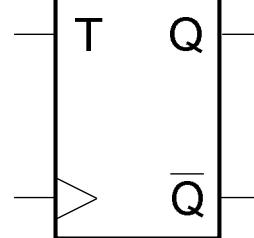
O circuito resultante também usa a variável de estado complementada, \overline{M}^* , como entrada do bloco FES, porque se consegue obter um circuito combinatório mais simples.



☐ Síntese do *flip-flop T* a partir do *flip-flop D*

Também é possível obter um *flip-flop T* a partir de um *D*, pelo mesmo processo seguido para obtenção do *J-K* a partir do *D*.

Neste caso, tem-se uma nova situação, na qual a FES depende da variável externa *T*, para além de depender da variável de estado.





☐ Síntese do $flip-flop T$ a partir do $flip-flop D$

É necessário ter em atenção o ASM do $flip-flop$ final e a tabela de transição de estados do $flip-flop$ na qual se baseia a síntese.

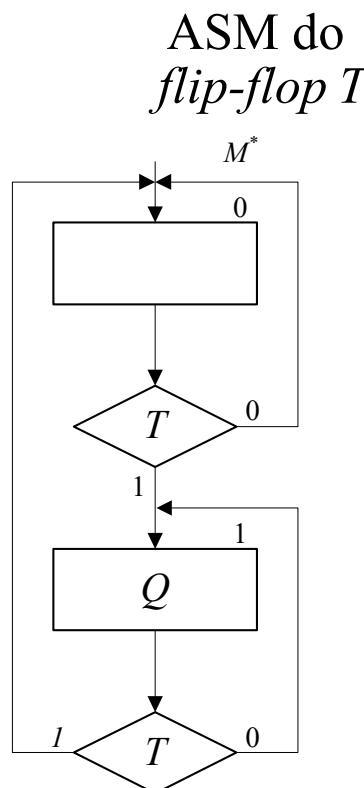


Tabela de transição de estados do $flip-flop T$:

Q^*	Q	T
0	0	0
0	1	1
1	0	1
1	1	0

Funções de estado seguinte e de saída

M^*	T	D	Q
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

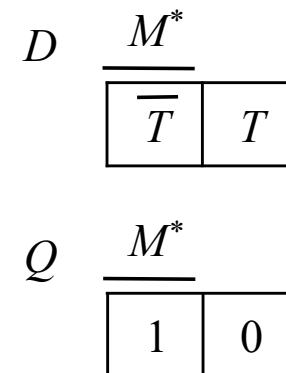


Síntese do $flip-flop T$ a partir do $flip-flop D$

A tabela contendo as funções de estado seguinte e de saída pode ser condensada, fazendo-se a inserção da variável de entrada T .

M^*	T	D	Q
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

M^*	D	Q
0	T	0
1	\bar{T}	1



$$D = \overline{M^*} \cdot T + M^* \cdot \bar{T} = \\ = M^* \oplus T$$

$$Q = M^*$$

O preenchimento do mapa de Karnaugh faz-se recorrendo à inserção das variáveis de entrada.

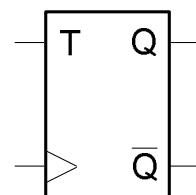
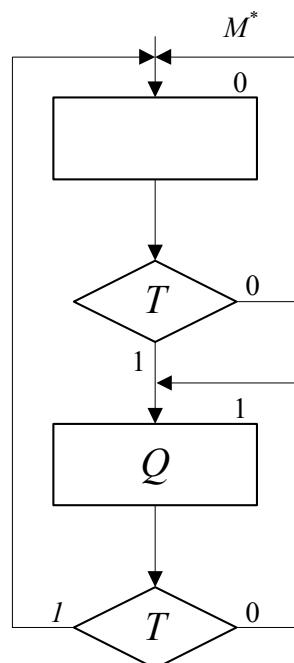
Como, no $flip-flop D$, existe identidade entre D e Q , escolhe-se o valor das variáveis que conduz ao valor de estado seguinte igual a 1.



☐ Síntese do *flip-flop T* a partir do *flip-flop D*

A obtenção de um *flip-flop T* a partir de um *D*, é levada a cabo segundo os mesmos princípios seguidos para obtenção do *J-K* a partir do *D*.

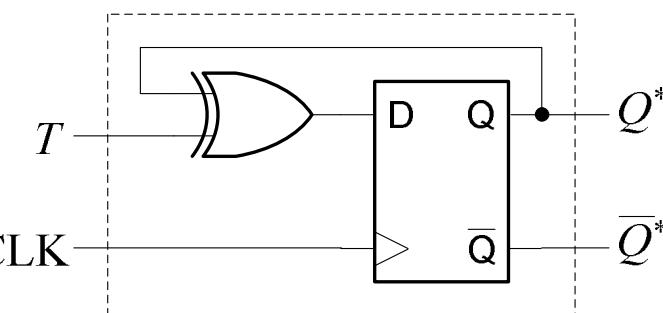
ASM do
flip-flop T



$$D = M^* \oplus T$$

$$Q = M^*$$

Síntese de um *flip-flop T* a partir de um *flip-flop D*



Tal como para o caso do *flip-flop J-K*, a função que fornece a saída *Q*, é uma identidade com a variável de estado M^* .



□ Projeto de um contador de módulo variável

Projetar um contador módulo 3 ou módulo 4, em função de uma entrada M :

- $M = 0$, contagem em módulo 3;
- $M = 1$, contagem em módulo 4.

Existe uma entrada S que para a contagem, mantendo o valor, ou deixa avançar a contagem:

- $S = 0$, avanço normal;
- $S = 1$, para a contagem no valor máximo de contagem.

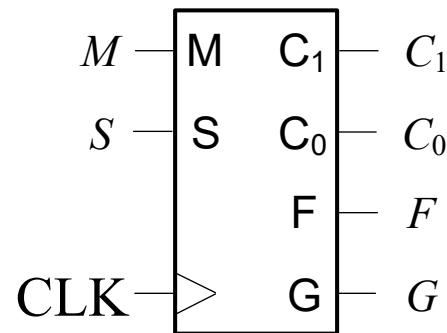
Tem quatro saídas:

- C_1 , saída de contagem de peso 1;
- C_0 , saída de contagem de peso 0;
- F , indicando que a configuração de contagem é ímpar;
- G , indicando que a configuração de contagem é ≥ 2 , quando em contagem em módulo 4.

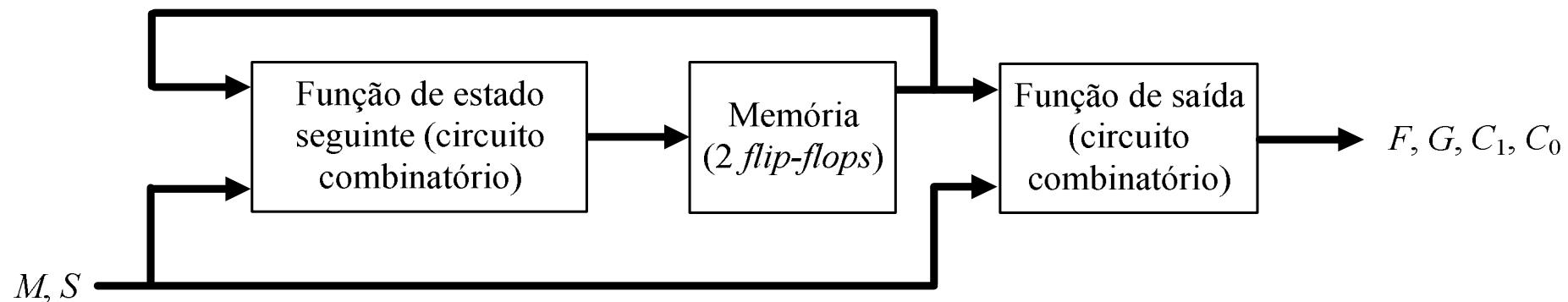


□ Projeto de um contador de módulo variável

Modelo “caixa preta” Aspetos especiais a ter em atenção:



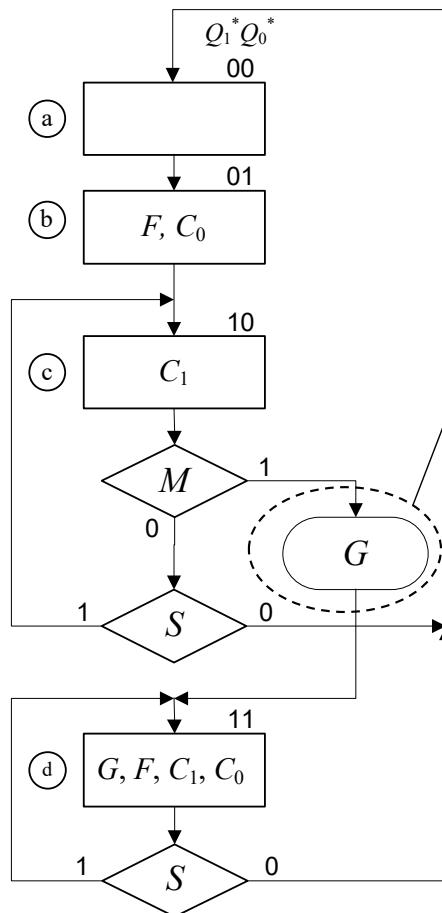
- As entradas M e S não são síncronas com o *clock*;
- As saídas F , C_1 e C_0 são síncronas com o *clock*;
- A saída G não é síncrona com o *clock* e é dependente não só do estado, mas também da entrada M .





□ Projeto de um contador de módulo variável

ASM do sistema



Mapa de *state assignment*:

$\underline{Q_0}^*$		Q_1^*
a	b	
c	d	

Saída função de estado e entrada. Significa que no estado 10, se $M = 1$, G fica ativada. No estado d, esta permanece ativa, mas se tal não acontecesse, G poderia ativar-se apenas momentaneamente (*glitch*). Nas presentes condições, G pode estar ativa até desde o início do estado c.

Síntese com *flip-flops D*

D_1	$\underline{Q_0}^*$		
	0	1	• Se M , fica 1
	S+M	S	• Se $\overline{M} \cdot S$, fica 1 • Se $\overline{M} \cdot \overline{S}$, fica 0

$\overline{M} \cdot S + M = S + M$

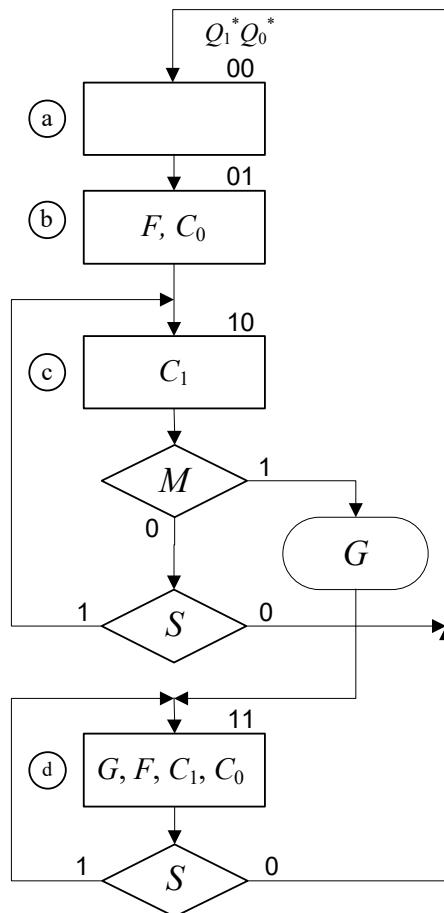
• Se S , fica 1
• Se \overline{S} , fica 0

$$D_1 = \overline{Q_1^*} \cdot Q_0^* + Q_1^* \cdot Q_0^* \cdot S + Q_1^* \cdot \overline{Q_0^*} \cdot (S + M) = \overline{Q_1^*} \cdot Q_0^* + Q_1^* \cdot S + Q_1^* \cdot \overline{Q_0^*} \cdot M$$



□ Projeto de um contador de módulo variável

ASM do sistema



Mapa de *state assignment*:

$\overline{Q_0}^*$		Q_1^*
a	b	
c	d	

Extração de expressões simplificadas em mapas de Karnaugh com variáveis inseridas:

- Extrair os “1”s presentes no mapa, considerando como “0” as variáveis, e aproveitando os eventuais *don't care* para simplificação;
- Extrair as variáveis inseridas, considerando também os “1”s como *don't care*.

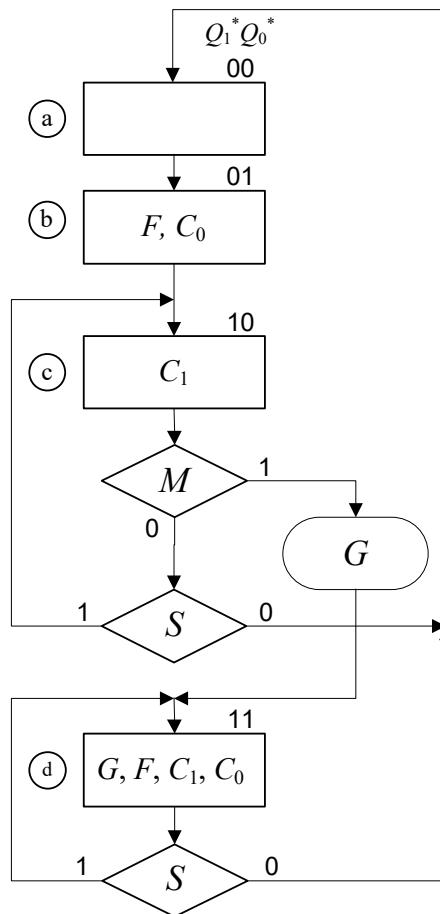
D_1		$\overline{Q_0}^*$	Q_1^*
0	1		
S+M	S		

$$\begin{aligned}D_1 &= \overline{Q_1^*} \cdot Q_0^* + Q_1^* \cdot Q_0^* \cdot S + Q_1^* \overline{Q_0^*} \cdot (S + M) = \\&= \overline{Q_1^*} \cdot Q_0^* + Q_1^* \cdot S + Q_1^* \cdot \overline{Q_0^*} \cdot M\end{aligned}$$



□ Projeto de um contador de módulo variável

ASM do sistema



Mapa de *state assignment*:

$\overline{Q_0}^*$		Q_1^*
a	b	
c	d	

D_0	$\overline{Q_0}^*$		Q_1^*
1	1	0	
M	M	S	Q_1^*

- Se M , fica 1
- Se $\overline{M} \cdot S$, fica 0
- Se $\overline{M} \cdot \overline{S}$, fica 0
- Se S , fica 1
- Se \overline{S} , fica 0

$$D_0 = \overline{Q_1^*} \cdot \overline{Q_0^*} + Q_1^* \cdot \overline{Q_0^*} \cdot M + Q_1^* \cdot Q_0^* \cdot S = \overline{Q_1^*} \cdot \overline{Q_0^*} + \overline{Q_0^*} \cdot M + Q_1^* \cdot Q_0^* \cdot S$$

Funções das variáveis de saída:

$$C_1 = Q_1^* \cdot \overline{Q_0^*} + Q_1^* \cdot Q_0^* = Q_1^*$$

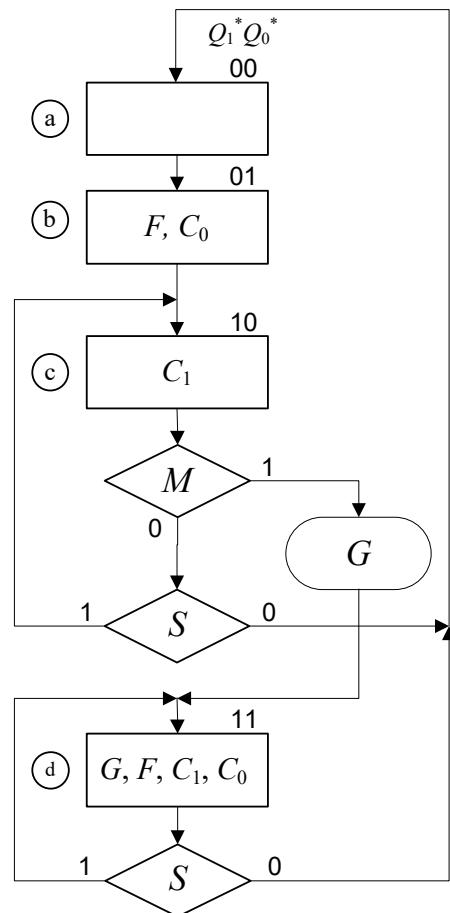
$$C_0 = \overline{Q_1^*} \cdot Q_0^* + Q_1^* \cdot Q_0^* = Q_0^*$$

$$F = \overline{Q_1^*} \cdot Q_0^* + Q_1^* \cdot Q_0^* = Q_0^*$$

$$G = Q_1^* \cdot \overline{Q_0^*} \cdot M + Q_1^* \cdot Q_0^* = Q_1^* \cdot (\overline{Q_0^*} \cdot M + Q_0^*) = Q_1^* \cdot (M + Q_0^*)$$

□ Projeto de um contador de módulo variável – FES com *flip-flops D*

ASM do sistema



Q_1^*	Q_0^*	M	S	Q_1	Q_0	D_1	D_0
0	0	0	0	0	1	0	1
0	0	0	1	0	1	0	1
0	0	1	0	0	1	0	1
0	0	1	1	0	1	0	1
0	1	0	0	1	0	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	0
1	0	0	0	0	0	0	0
1	0	0	1	1	0	1	0
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	1
1	1	0	0	0	0	0	0
1	1	0	1	1	1	1	1
1	1	1	0	0	0	0	0
1	1	1	1	1	1	1	1

D_1	M			
Q_1^*	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	0	0	1	1
1	1	1	1	1

$$D_1 = \overline{Q_1^*}Q_0^* + Q_1^*.S + Q_1^*.Q_0^*.M$$

D_0	M			
Q_1^*	1	1	1	1
0	1	1	1	0
0	0	1	1	1
0	0	0	1	1
0	0	0	0	0

$$D_0 = \overline{Q_1^*}\overline{Q_0^*} + \overline{Q_0^*}.M + Q_1^*.Q_0^*.S$$



□ Projeto de um contador de módulo variável - FS

Q_1^*	Q_0^*	M	S	C_1	C_0	F	G
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	1	0	1	1	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1
1	1	0	0	1	1	1	1
1	1	0	1	1	1	1	1
1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1

C_1	M			
Q_1^*	0	0	0	0
0	1	1	1	1
1	1	1	1	1
0	0	0	0	0

$$C_1 = Q_1^*$$

F	M			
Q_1^*	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	1	1

$$F = Q_0^*$$

C_0	M			
Q_1^*	0	0	0	0
0	0	0	0	0
1	1	1	1	1
1	1	1	1	1

$$C_0 = Q_0^*$$

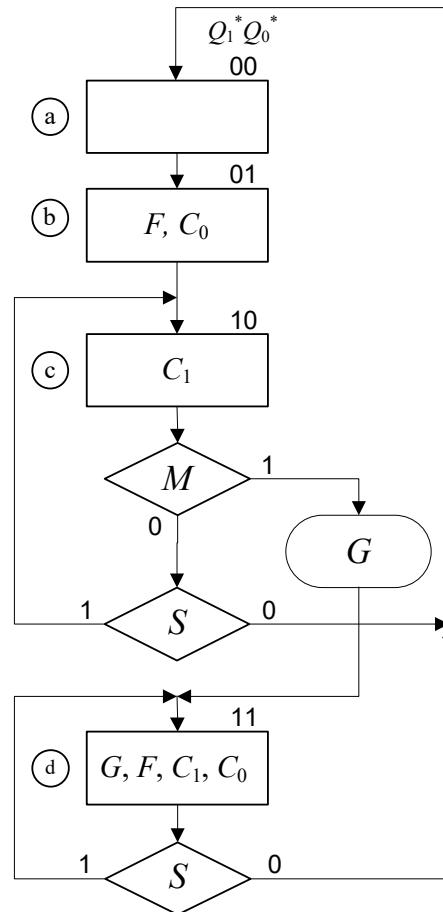
G	M			
Q_1^*	0	0	0	0
0	1	1	1	0
1	1	1	1	1
0	0	0	0	0

$$G = Q_1^*.M + Q_1^*.Q_0^* = Q_1^*.(M + Q_0^*)$$



□ Projeto de um contador de módulo variável

ASM do sistema

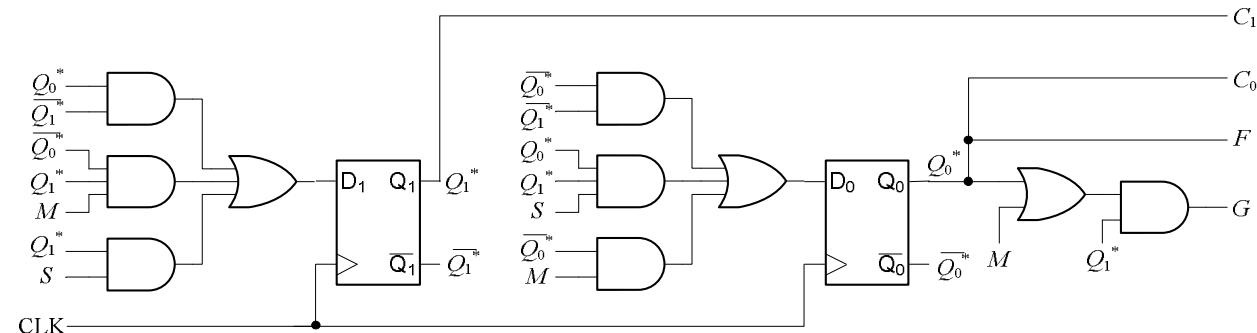
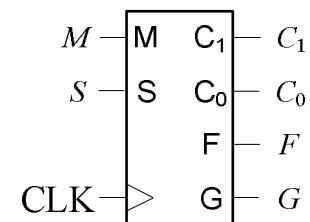


Síntese com *flip-flops D*

Funções de estado $D_1 = \overline{Q_1^*}Q_0^* + Q_1^*S + Q_1^*\overline{Q_0^*}M$
seguinte: $D_0 = \overline{Q_1^*}\overline{Q_0^*} + \overline{Q_0^*}M + Q_1^*Q_0^*S$

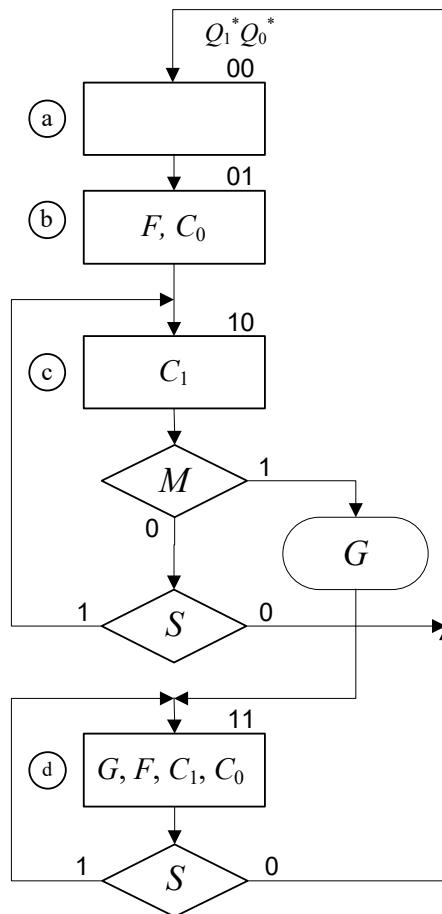
Funções de saída: $C_1 = Q_1^*$
 $C_0 = Q_0^*$

$F = Q_0^*$
 $G = Q_1^*(M + Q_0^*)$



□ Projeto de um contador de módulo variável

ASM do sistema



Mapa de *state assignment*:

$\overline{Q_0}^*$		Q_1^*
a	b	
c	d	

Síntese com *flip-flops J-K*

J_1	$\overline{Q_0}^*$		Q_1^*
	0	1	
	-	-	

$$J_1 = Q_0^*$$

K_1	$\overline{Q_0}^*$		Q_1^*
	-	-	
	$\bar{M} \cdot \bar{S}$	\bar{S}	

- Se S , fica 1
- Se \bar{S} , fica 0
- Se M , fica 1
- Se $\bar{M} \cdot \bar{S}$, fica 1
- Se $\bar{M} \cdot \bar{S}$, fica 0

Q^*	Q	J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

- Se S , fica 1
- Se \bar{S} , fica 0

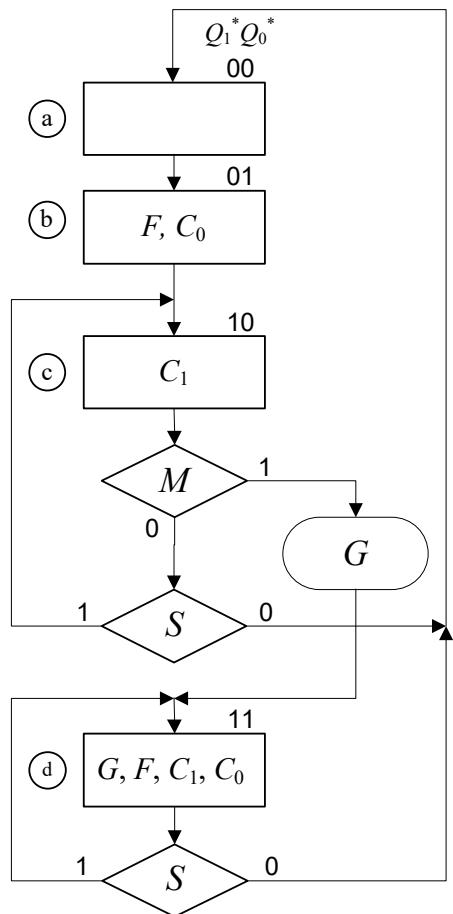
Neste estado, as transições possíveis são $1 \rightarrow 1$ ou $1 \rightarrow 0$. Pela tabela de transição, vê-se que K fica com a negação de Q , por isso, vai pelo “0”.

$$K_1 = Q_0^* \cdot \bar{S} + \overline{Q_0^*} \cdot \bar{M} \cdot \bar{S} = \bar{S} \cdot (Q_0^* + \bar{M})$$



□ Projeto de um contador de módulo variável

ASM do sistema



Mapa de *state assignment*:

$\overline{Q_0}^*$		Q_1^*	
a	b	c	d

Síntese com *flip-flops J-K*

J_0	$\overline{Q_0}^*$		Q_1^*
	1	-	
	M	-	

- Se M , fica 1
- Se $\overline{M} \cdot S$, fica 0
- Se $\overline{M} \cdot \overline{S}$, fica 0

$$J_0 = \overline{Q_1}^* + M$$

Q^*	Q	J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

K_0	$\overline{Q_0}^*$		Q_1^*
	-	1	
	-	\overline{S}	

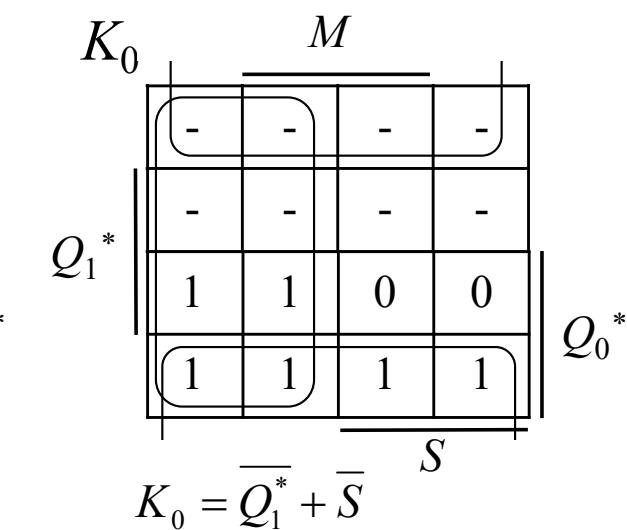
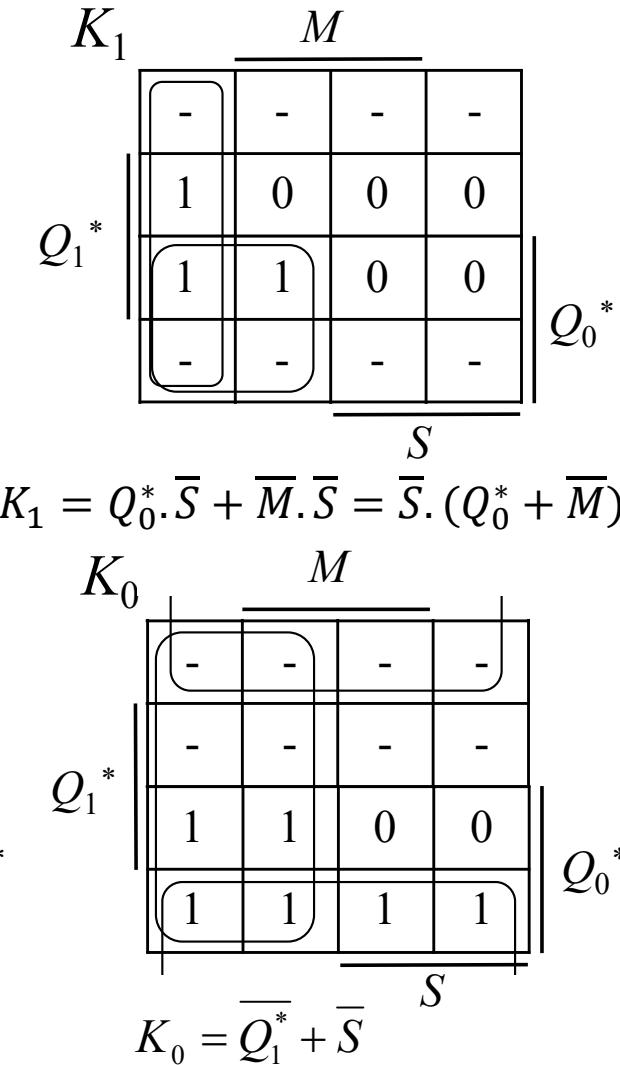
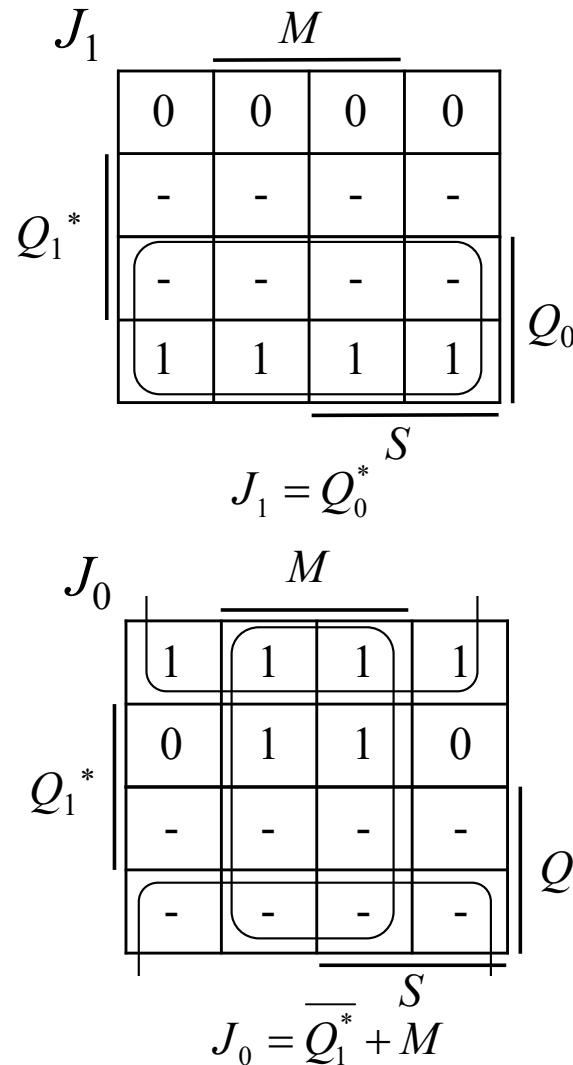
- Se S , fica 1
- Se \overline{S} , fica 0

$$K_0 = \overline{Q_1}^* + \overline{S}$$



□ Projeto de um contador de módulo variável – FES com *flip-flops J-K*

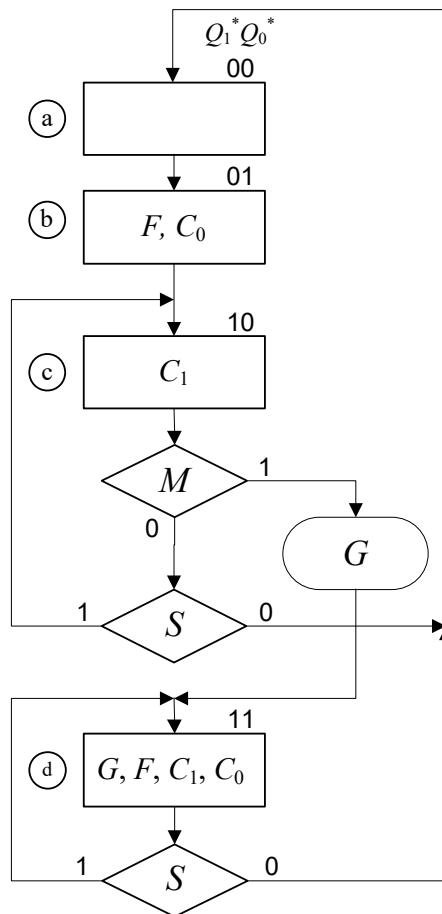
Q_1^*	Q_0^*	M	S	Q_1	Q_0	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	-	1	-
0	0	0	1	0	1	0	-	1	-
0	0	1	0	0	1	0	-	1	-
0	0	1	1	0	1	0	-	1	-
0	1	0	0	1	0	1	-	-	1
0	1	0	1	1	0	1	-	-	1
0	1	1	0	1	0	1	-	-	1
0	1	1	1	1	0	1	-	-	1
1	0	0	0	0	0	-	1	0	-
1	0	0	1	1	0	-	0	0	-
1	0	1	0	1	1	-	0	1	-
1	0	1	1	1	1	-	0	1	-
1	1	0	0	0	0	-	1	-	1
1	1	0	1	1	1	-	0	-	0
1	1	1	0	0	0	-	1	-	1
1	1	1	1	1	1	-	0	-	0





□ Projeto de um contador de módulo variável

ASM do sistema

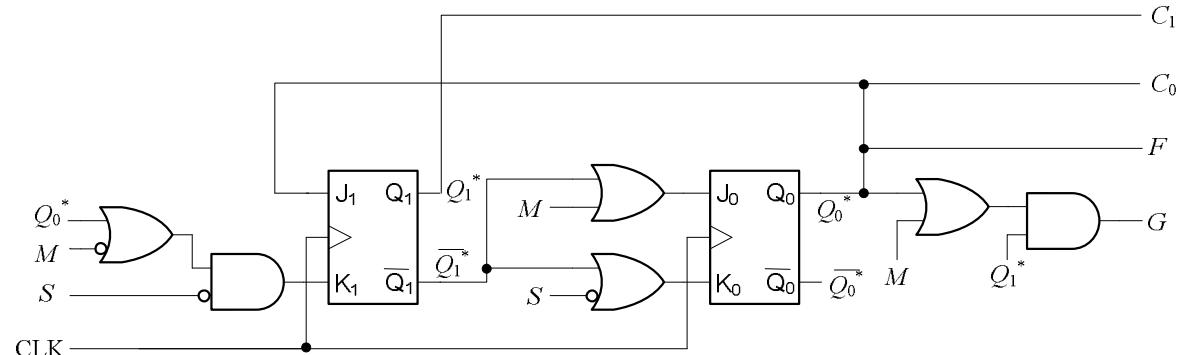
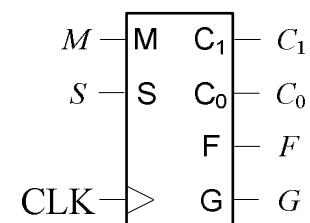


Síntese com *flip-flops J-K*

Funções de estado
seguinte:

$$\begin{aligned}J_1 &= Q_0^* \\K_1 &= \bar{S}.(Q_0^* + \bar{M}) \\J_0 &= \bar{Q}_1^* + M \\K_0 &= \bar{Q}_1^* + \bar{S}\end{aligned}$$

As funções de saída são as mesmas que para a implementação com *flip-flops D*.





□ Simulação da operação de *flip-flops* no Arduino

- É possível simular a operação de *flip-flops* no Arduino.
- Apesar de serem estruturas de *hardware* existentes em circuito integrado, é possível estabelecer em *software* o mecanismo através do qual se pode simular a operação de cada um dos tipos de *flip-flop* já abordados.
- Dar-se-á particular ênfase aos *flips-flops edge-triggered*, dado estes se adequarem melhor ao tipo de circuitos que se pretende desenvolver.
- Como a atualização do estado dos *flip-flops* depende da existência de um *clock*, este terá que ser gerado externamente.



☐ “Interrupções” no Arduino

- A transição ativa de *clock* origina a resposta de dado tipo de *flip-flop*.
- Como esta transição do sinal de *clock* é um evento externo, tem que se lidar com ele recorrendo a interrupções (*interrupts*).
- Os *interrupts* são úteis quando se pretende ter uma dada resposta do programa em Arduino à ocorrência de algum estímulo externo proveniente de algum dispositivo ou do próprio utilizador.
- Se não se utilizassem *interrupts*, o programa teria continuamente que verificar a ocorrência desse estímulo, realizando *polling*.



□ “Interrupções” no Arduino

- O *interrupt* consiste em desviar a execução do programa, de modo a ser executada uma função específica. Quando esta termina, volta-se à execução original, no ponto onde esta tinha ficado antes de acontecer a interrupção.
- Função que permite associar um *interrupt* ao Arduino, no `setup()`:

```
attachInterrupt(interrupt, function, mode);
```

Parâmetros: `interrupt` – n.º do *interrupt* [0 (pino 2) ou 1 (pino 3)]

`function` – Nome da função a ser executada no *interrupt*

`mode` – Modo de sensibilidade ao evento físico de *interrupt*



□ Parâmetros da função `attachInterrupt()` no Arduino

- interrupt

São possíveis dois *interrupts* físicos no Arduino, 0 ou 1, atribuídos aos pinos digitais 2 ou 3, respetivamente.

Pode usar-se a macro `digitalPinToInterrupt(pin)` por forma a mapear o pino digital para o número do *interrupt* correspondente.

O *interrupt* 0 é mais prioritário do que o *interrupt* 1.



□ Parâmetros da função `attachInterrupt()` no Arduino

- `function`

Função a ser executada quando o *interrupt* físico ocorre. Também se designa por ISR (*Interrupt Service Routine*).

Esta função não pode ter argumentos e não deverá devolver valor algum.

Deve ser uma função o mais eficiente possível, do ponto de vista computacional, de modo a demorar o menor tempo possível a ser executada.

As variáveis globais que forem manipuladas devem ter o qualificador `volatile`, por forma a garantir que sejam guardadas em memória principal.



□ Parâmetros da função `attachInterrupt()` no Arduino

- mode

Define o tipo de evento acontece no pino de *interrupt*. São usadas quatro constantes para definir os modos de desencadeamento do *interrupt*:

LOW - Quando o pino está a “0”;

CHANGE - Quando o pino muda de valor (de “1” para “0” e vice-versa);

RISING - Quando o pino muda de “0” para “1”;

FALLING - Quando o pino muda de “1” para “0”.



□ Parâmetros da função `attachInterrupt()` no Arduino

Após se estabelecer a *Interrupt Service Routine* e, em que circunstâncias esta deve ser executada, deve evocar-se a função `attachInterrupt()` na função `setup()`, para que o Arduino seja sensível a uma dada interrupção.

Exemplo:

```
void setup() {  
  
    ...  
  
    attachInterrupt(0, ISR, RISING);  
  
}
```

Para desligar uma dada interrupção, deve usar-se a função `detachInterrupt(interrupt)`.



☐ Ligar ou desligar a totalidade das interrupções no Arduino

De maneira a, de uma forma geral, se poder ligar ou desligar os *interrupts* no Arduino, deve usar-se, respetivamente:

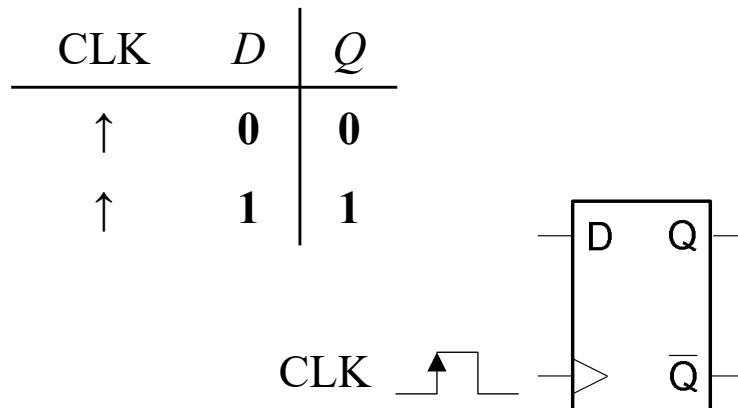
```
interrupts(); // Ligar as interrupções no Arduino  
  
noInterrupts(); // Desligar as interrupções no Arduino
```

`noInterrupts()` utiliza-se em certas zonas nas quais o código a executar é crítico em termos de tempo, não devendo ser interrompido.



□ Simulação da operação de *flip-flops D* no Arduino

- Usando as interrupções, a memorização de informação por um *flip-flop* será realizada no momento da transição dos impulsos de *clock*, tipicamente, na transição ascendente. No *flip-flop D*, a função de simulação do *flip-flop* retorna um valor binário que é função só da entrada.



```
boolean D;           // Entrada D
volatile boolean Q; // Saída Q

boolean flip_flop_D(boolean D) {
    return D;
}

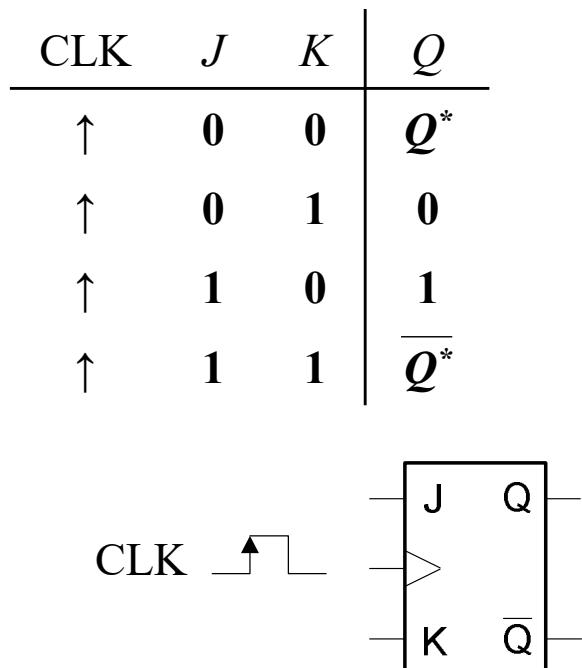
void CLK() {
    Q = flip_flop_D(D);
}

void setup() {
    attachInterrupt(0, CLK, RISING);
    interrupts();
}
```



□ Simulação da operação de *flip-flops J-K* no Arduino

- A simulação dos *flip-flops J-K* também é feita pela sua definição. Para estes *flip-flops*, a sua função de simulação retorna um valor binário que é função do estado presente e das entradas.



```
boolean J, K; // Entradas J e K
volatile boolean Q; // Saída Q

boolean flip_flop_JK(boolean Q, boolean J, boolean K) {
    return Q & !K | !Q & J;
}

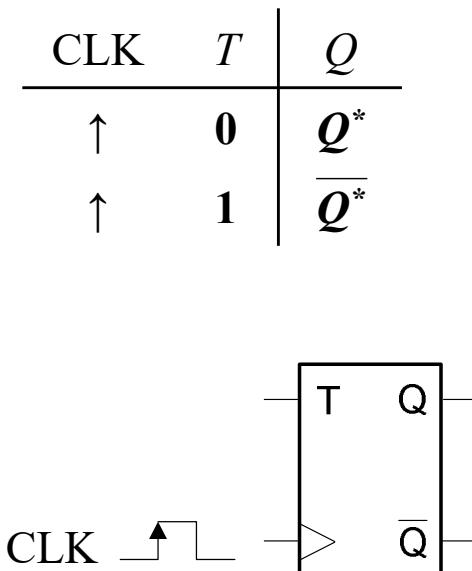
void CLK(){
    Q = flip_flop_JK(Q, J, K);
}

void setup(){
    attachInterrupt(0, CLK, RISING);
    interrupts();
}
```



□ Simulação da operação de *flip-flops T* no Arduino

- A simulação dos *flip-flops T* também é feita pela sua definição. Para estes *flip-flops*, a sua função de simulação retorna um valor binário que é função do estado presente e das entradas.



```
boolean T;           // Entrada T
volatile boolean Q; // Saída Q

boolean flip_flop_T(boolean Q, boolean T) {
    return Q ^ T;
}

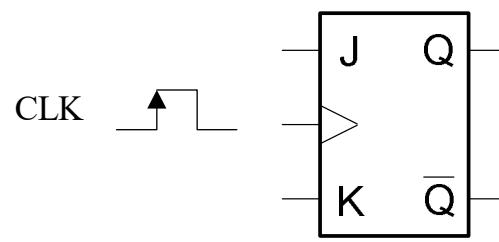
void CLK(){
    Q = flip_flop_T(Q, T);
}

void setup(){
    attachInterrupt(0, CLK, RISING);
    interrupts();
}
```

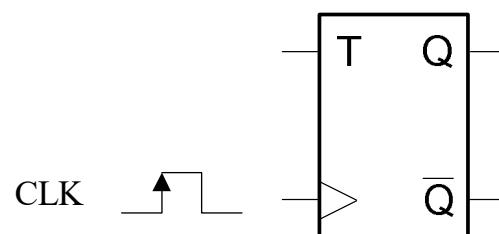


□ Simulação da operação de *flip-flops* no Arduino

- Em alternativa, a simulação dos *flip-flops* *J-K* e *T* pode ser feita funcionalmente, recorrendo à interpretação literal da definição narrativa de cada um desses tipos de *flip-flop*



```
boolean J, K;  
volatile boolean Q;  
  
boolean flip_flop_JK(boolean Q, boolean J, boolean K) {  
    return Q ? (K ? 0 : 1) : (J ? 1 : 0);  
}
```

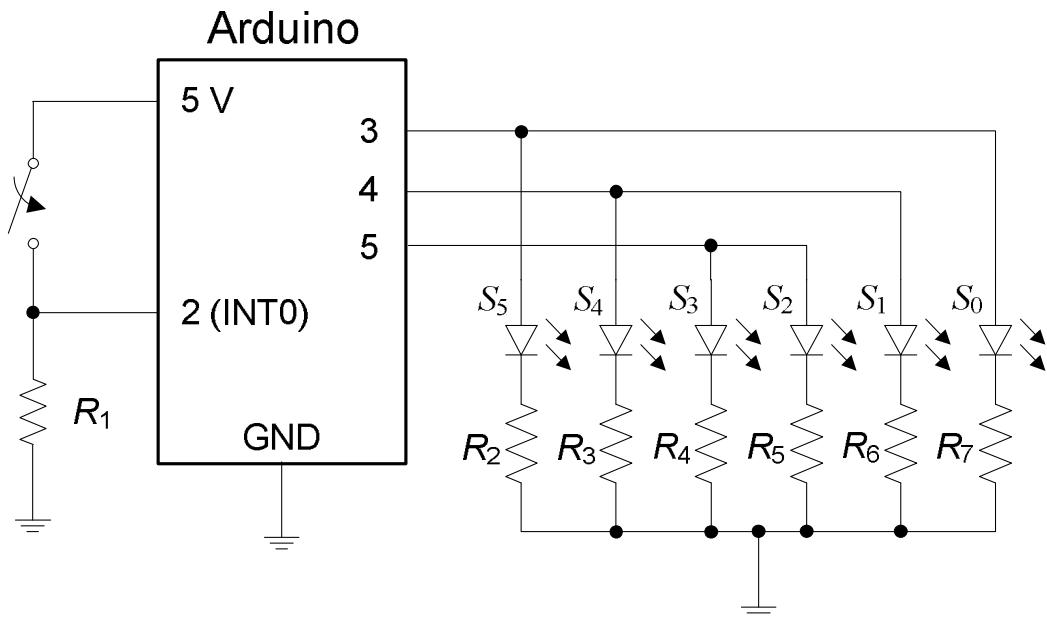


```
boolean T;  
volatile boolean Q;  
  
boolean flip_flop_T(boolean Q, boolean T) {  
    return T ? !Q : Q;  
}
```



☐ Exemplo de implementação do gerador de padrões sequenciais

Esquema elétrico do circuito:



Os impulsos de CLK são dados através de *clicks* num botão de pressão, de modo a gerar as transições ascendentes.

As saídas S_5 a S_0 são constituídas, cada uma, por um LED e uma resistência em série.



☐ Exemplo de implementação do gerador de padrões sequenciais (cont.)

Esquema de atribuição de pinos e declaração de variáveis globais.

Assume-se uma implementação baseada em *flip-flops D*.

```
// Atribuição de pinos à entrada de CLK e às saídas para os LEDs
#define pino_CLK      2 // INT0
#define pino_S5_S0    3
#define pino_S4_S1    4
#define pino_S3_S2    5

// Entradas dos flip-flops
boolean D1, D0;

// Saídas dos flip-flops
volatile boolean Q1, Q0;

// Valores das saídas do sistema
boolean S5_S0, S4_S1, S3_S2;
```



☐ Exemplo de implementação do gerador de padrões sequenciais (cont.)

Implementação dos módulos combinatórios de geração de estado seguinte e de funções de saída:

```
// Bloco combinatório de geração de estado seguinte
void funcoesEstadoSeguinte() {
    D1 = Q1 ^ Q0;
    D0 = !Q0;
}
```

$$D_1 = Q_1^* \oplus Q_0^*$$
$$D_0 = \overline{Q_0^*}$$

```
// Bloco combinatório de geração das funções de saída
void funcoesSaida() {
    S5_S0 = !Q1 & !Q0;
    S4_S1 = Q0;
    S3_S2 = Q1 & !Q0;
}
```

$$S_5 = S_0 = \overline{Q_1^*} \overline{Q_0^*}$$
$$S_4 = S_1 = Q_0^*$$
$$S_3 = S_2 = Q_1^* \overline{Q_0^*}$$



☐ Exemplo de implementação do gerador de padrões sequenciais (cont.)

Implementação das saídas e dos *flip-flops*.

Sobre as saídas, tem que se ter atenção a configurar os pinos pretendidos como OUTPUT, na função `setup()`. A função de *clock* (CLK) deve ser indicada na função `attachInterrupt()` e deve simular os *flip-flops*.

```
void escreveSaídas() {
    digitalWrite(pino_S5_S0, S5_S0);
    digitalWrite(pino_S4_S1, S4_S1);
    digitalWrite(pino_S3_S2, S3_S2);
}

boolean flip_flop_D(boolean D) {
    return D;
}

void CLK() {
    Q1 = flip_flop_D(D1);
    Q0 = flip_flop_D(D0);
}
```



☐ Exemplo de implementação do gerador de padrões sequenciais (cont.)

Funções fundamentais do Arduino, `setup()` e `loop()`.

Sobre as saídas, tem que se ter atenção a configurar os pinos pretendidos como `OUTPUT`, na função `setup()`. A função que implementa os *flip-flops* deve ser chamada na função `CLK()`, indicada no `attachInterrupt()`. As funções combinatórias são executadas permanentemente no `loop()`.

```
void setup() {
    pinMode(pino_S5_S0, OUTPUT);
    pinMode(pino_S4_S1, OUTPUT);
    pinMode(pino_S3_S2, OUTPUT);
    attachInterrupt(digitalPinToInterrupt(pino_CLK), CLK, RISING);
    interrupts();
}

void loop() {
    funcoesEstadoSeguinte();
    funcoesSaida();
    escreveSaidas();
}
```



□ Exemplo de implementação do gerador de padrões sequenciais (cont.)

Em substituição do botão de pressão, pode ter-se um oscilador de onda quadrada a gerar o *clock* do circuito, sendo que o pino de saída desse oscilador deverá ligar ao pino de *interrupt* escolhido para o funcionamento.

Mostra-se de seguida uma possibilidade de função simples para realizar um gerador de *clock* com um *duty-cycle* de 50%, podendo definir-se o pino de saída e a frequência de operação (até ao máximo de 500 Hz). Na função *setup()*, tem que se configurar esse pino de saída como OUTPUT.

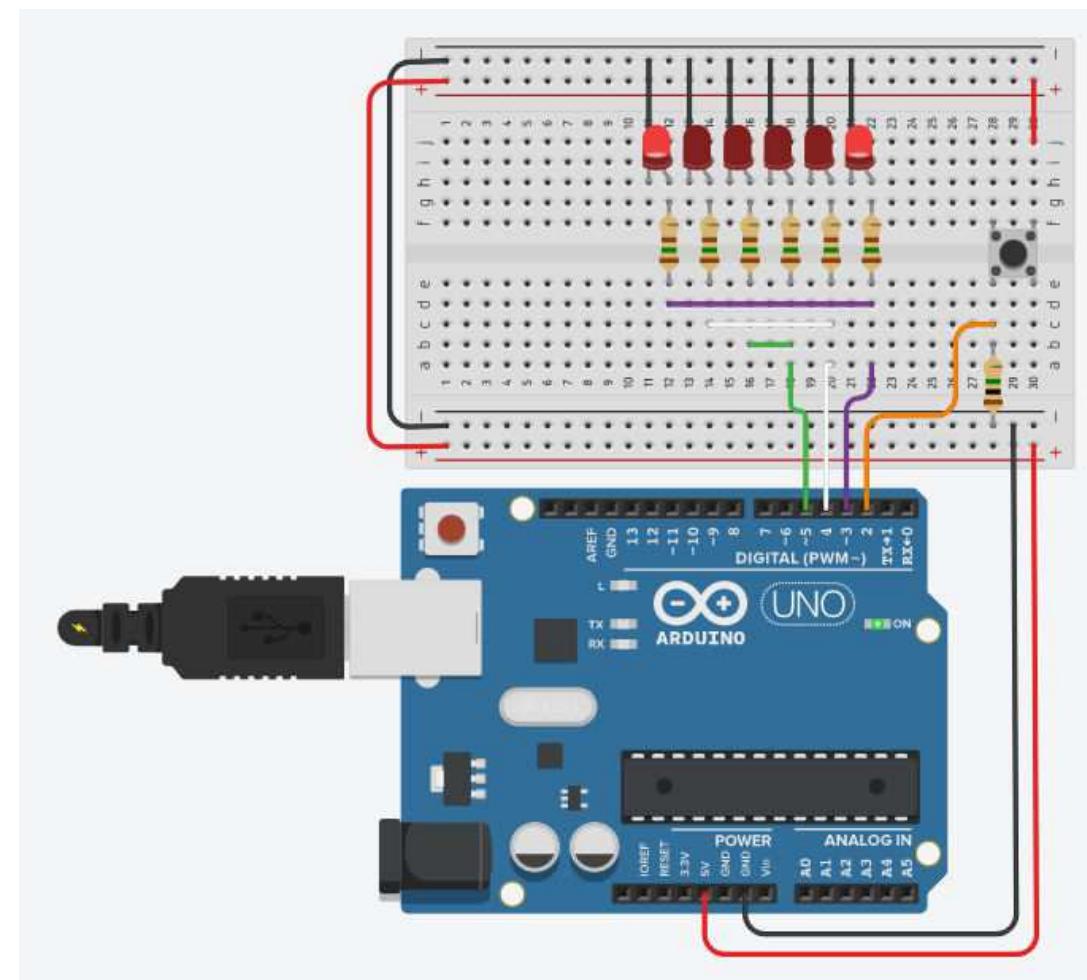
```
void clockGenerator(byte pino, float f){  
    static boolean state = LOW;  
    static unsigned long t1, t0;  
  
    t1 = millis();  
    if (t1 - t0 >= 500. / f){  
        digitalWrite(pino, state = !state);  
        t0 = t1;  
    }  
}
```

No caso de se usar esta função de geração de *clock*, a mesma deve ser evocada no *loop()*, tal como as funções combinatórias e de saída.

□ Exemplo de implementação do gerador de padrões sequenciais (cont.)

Simulado no Autodesk® TinkerCad®, usando o Arduino, fica-se com o circuito mostrado ao lado.

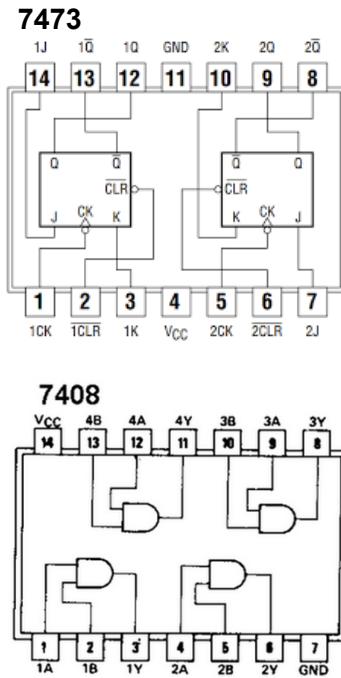
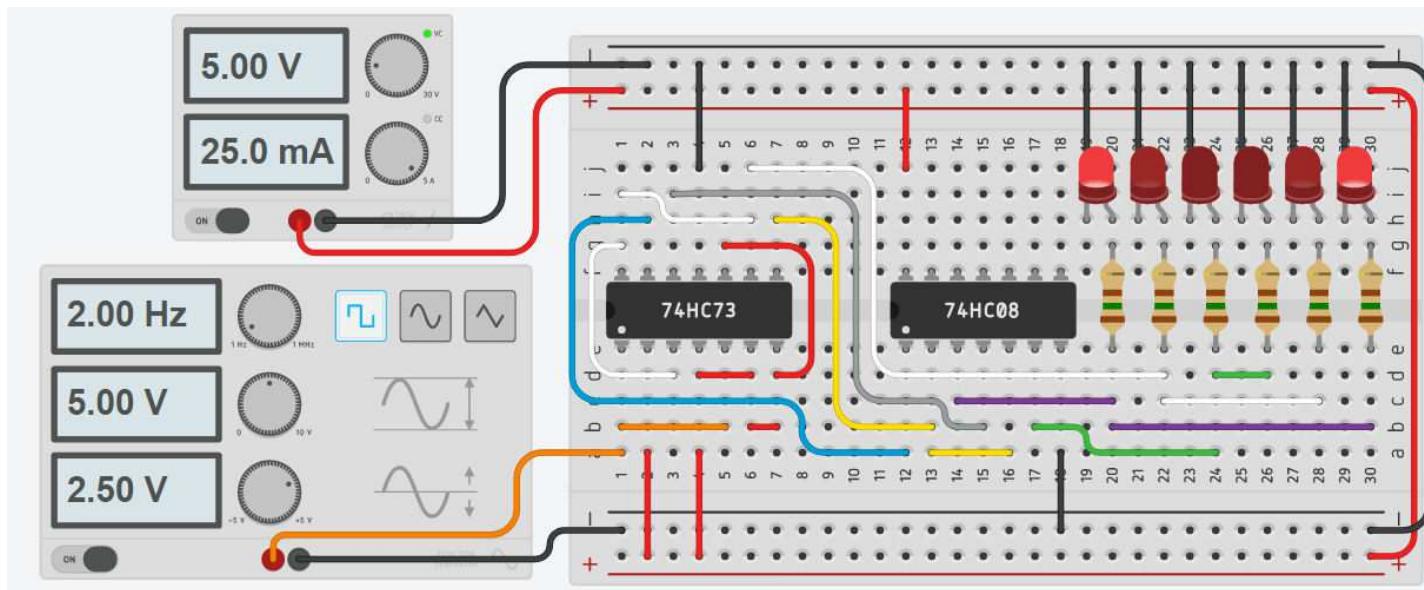
O *clock* é dado através do botão de pressão, em que, cada *click* provoca uma transição ascendente no pino 2 do Arduino, ou seja, provocará a chamada à função de atendimento da interrupção.





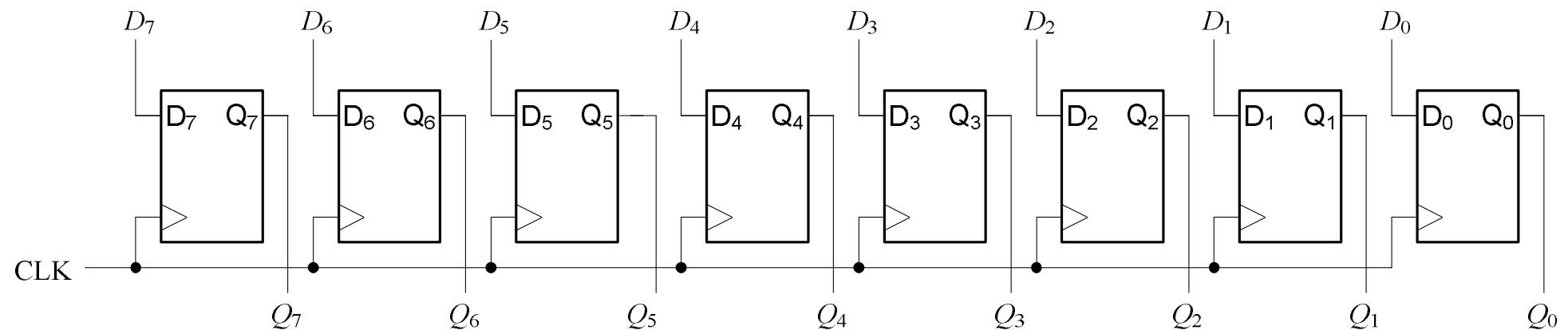
☐ Exemplo de implementação do gerador de padrões sequenciais (cont.)

É de realçar que se está a simular *hardware*. O circuito, realizado com circuitos integrados com *flip-flops J-K* e portas lógicas AND, seria como se mostra de seguida, também no Autodesk® TinkerCad®. A alimentação é dada por uma fonte de tensão e o *clock* por um gerador de sinais.



☐ Registros *edge-triggered*

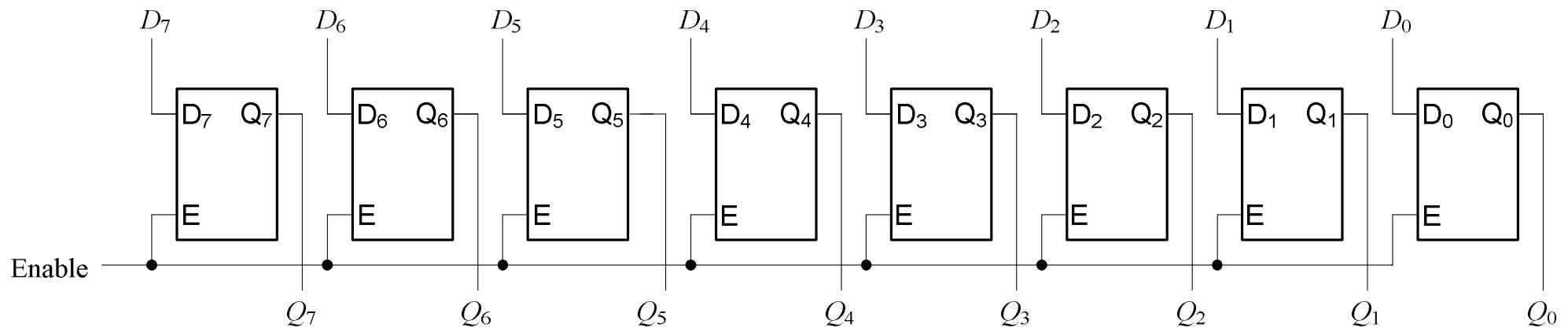
- Um registo é um conjunto de *flip-flops* destinado a guardar informação. Por exemplo, um registo *edge-triggered* destinado a guardar um *byte* de informação tem a seguinte configuração:



O valor do byte em D , será memorizado em Q no instante da transição ascendente de *clock*.

☐ Registros *latch*

- Também é possível terem-se registos do tipo *latch*, nos quais a memorização da informação é feita no momento em que o sinal de *enable* passa de ativo a inativo. Tal registo tem a configuração mostrada de seguida:



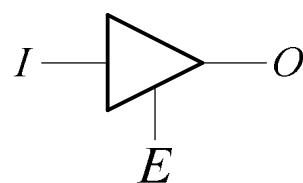
O valor do byte em D , será memorizado em Q no instante da transição descendente de *Enable*.



□ Portas *tri-state*

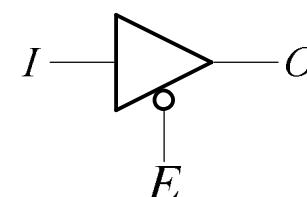
Tal como o nome indica, este tipo de portas possui três estados possíveis: “0”, “1” e “alta impedância” (resistência infinita – circuito aberto), significativo de que a porta se encontra desligada do resto do circuito que tenha a jusante da saída.

Existem algumas variantes possíveis, com o seu respetivo símbolo lógico:



Porta *tri-state* identidade
com controlo *active-high*

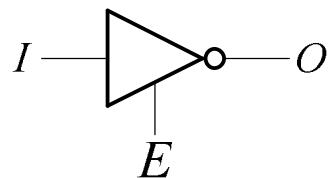
E	I	O
0	0	Z
0	1	Z
1	0	0
1	1	1



Porta *tri-state* identidade
com controlo *active-low*

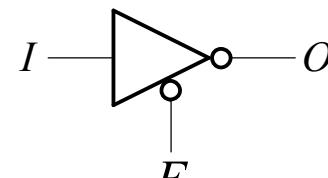
E	I	O
0	0	0
0	1	1
1	0	Z
1	1	Z

□ Portas *tri-state*



Porta *tri-state* inversora
com controlo *active-high*

<i>E</i>	<i>I</i>	<i>O</i>
0	0	Z
0	1	Z
1	0	1
1	1	0



Porta *tri-state* inversora
com controlo *active-low*

<i>E</i>	<i>I</i>	<i>O</i>
0	0	1
0	1	0
1	0	Z
1	1	Z

O terminal E (*enable*) permite que a porta esteja ativa (inserida no circuito), colocando na sua saída o valor que lhe for solicitado, se este sinal estiver ativo (*high* ou *low*). Caso o *enable* não esteja ativo, a saída fica “flutuante”, ou seja, como que desligada fisicamente. O conceito de *enable* é muito comum em sistemas digitais, determinando se um determinado dispositivo ou módulo se encontrará em atividade ou não.



□ Multiplexer 4x1 realizado com portas tri-state

Pode-se ter um *multiplexer* utilizando portas *tri-state*, fazendo recurso a um *decoder* para ativar uma e só uma das portas *tri-state*.

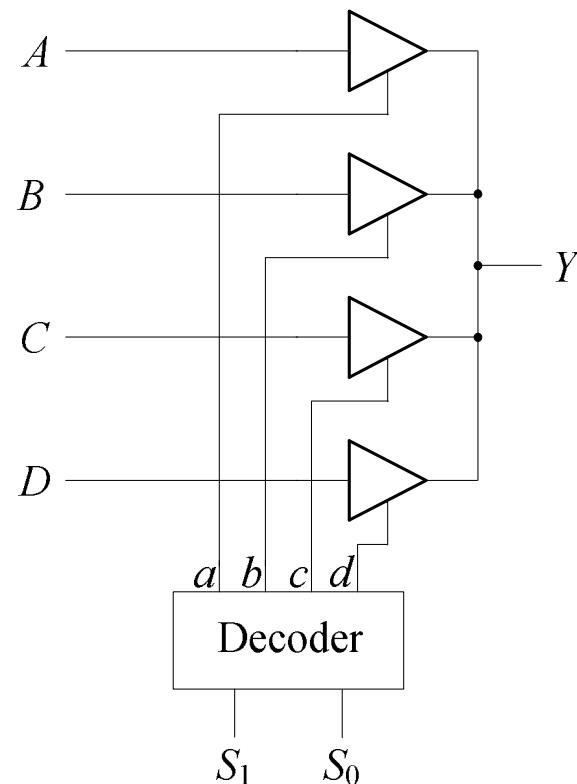
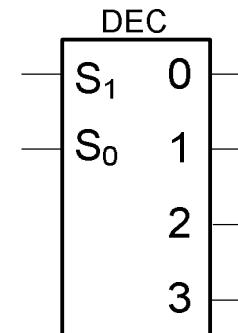


Tabela de verdade e expressões do *decoder*

S ₁	S ₀	a	b	c	d
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

$a = \overline{S_1} \cdot S_0$
 $b = \overline{S_1} \cdot \overline{S_0}$
 $c = S_1 \cdot \overline{S_0}$
 $d = S_1 \cdot S_0$

Símbolo lógico





☐ Registos bidirecionais e tipos de memória

A porta *tri-state* é um componente fundamental para se poder ter acesso a cada um dos *flip-flops* de um registo por via de uma linha bidirecional (*I/O*).

Com registos bidirecionais podem ter-se estruturas maiores que podem aglomerar milhões de registos, naquilo que conhecemos por memórias.

A estrutura interna das memórias comprehende não só os registos, mas também alguns módulos combinatórios já conhecidos, como o *demultiplexer*.

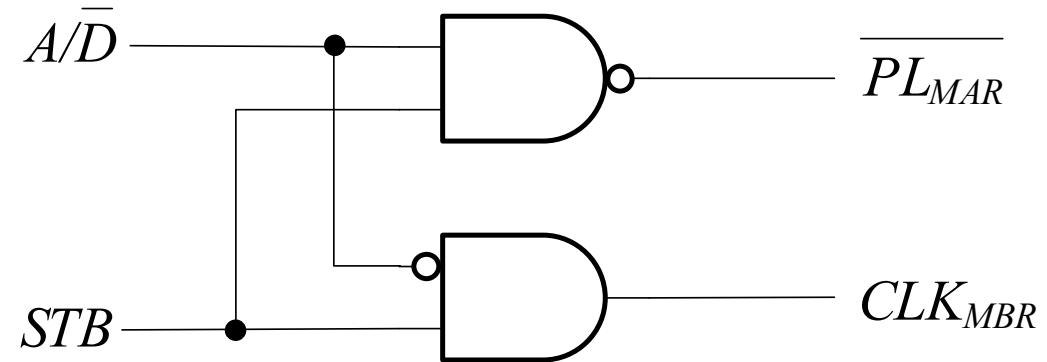
Existem memórias voláteis, que perdem os valores memorizados quando se desliga a alimentação (RAM) e outras que mantém a informação (ROM).



□ Sistema incluindo um MAR e um MBR

Respondendo às especificações dadas para a utilização de um *memory address register* (MAR) e de um *memory buffer register* (MBR), para construir um sistema de acesso a uma RAM, o projeto do circuito combinatório que fornece $\overline{PL_{MAR}}$, e CLK_{MBR} é o seguinte:

A/\bar{D}	STB	$\overline{PL_{MAR}}$	CLK_{MBR}
0	0	1	0
0	1	1	1
1	0	1	0
1	1	0	0



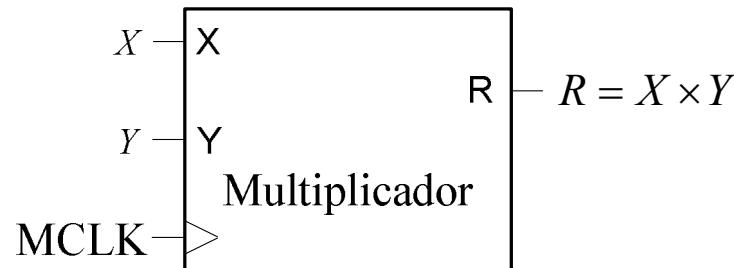


Microprocessadores



☐ Multiplicador por somas sucessivas

1. Diagrama de blocos, com as entradas e saídas do sistema



2. Algoritmo a ser cumprido

```
for (R = 0, I = Y; I != 0; --I)  
    R += X;
```

- R é inicializado a 0
- I é inicializado com o valor de Y
- Por cada iteração, R acumula a soma do seu valor com o de X , decrementando I
- Enquanto $I \neq 0$, a soma continua a realizar-se



☐ Multiplicador por somas sucessivas

3. Tipos de *hardware*

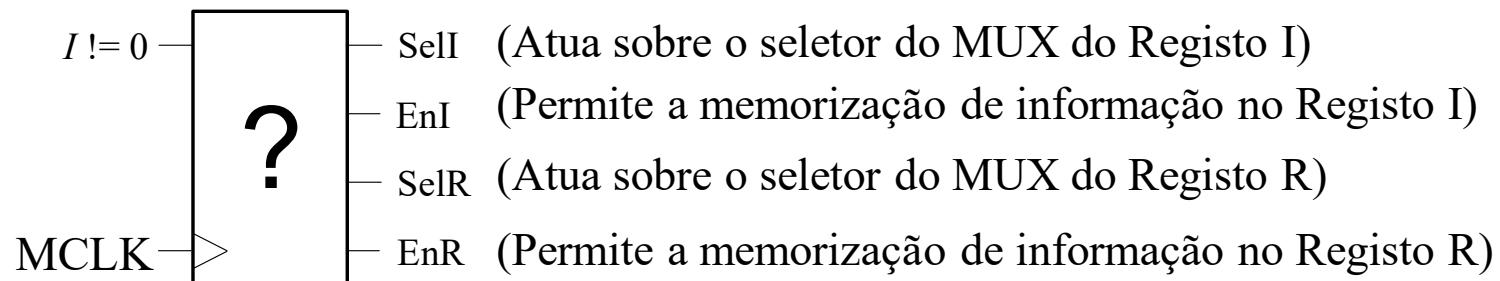
I : registo	Variável de iteração
R : registo	Variável de resultado
X : entrada	Operando
Y : entrada	Operando

4. Diagrama de blocos do módulo funcional

- Notar a existência de $\overline{\text{MCLK}}$: primeiro, as saídas do módulo de controlo ficam corretamente estabelecidas e só depois têm efeito no módulo funcional ($T_{\text{MCLK}}/2$ mais tarde).

☐ Multiplicador por somas sucessivas

5. Entradas e saídas do módulo de controlo



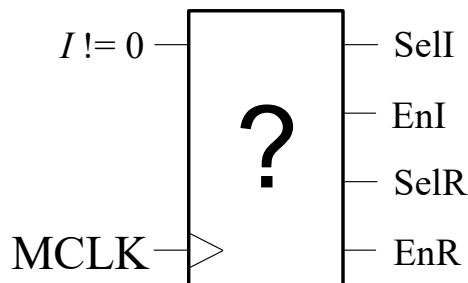
6. Projeto do módulo de controlo

Dois estados: 1 *flip-flop* (1.^º estado: inicialização, 2.^º estado: execução)

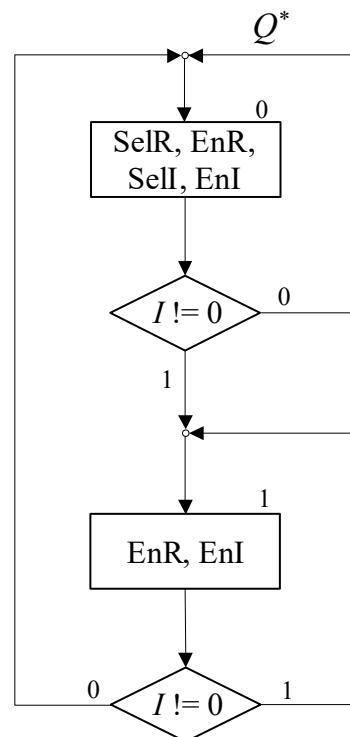
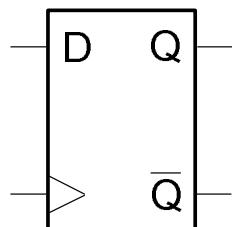
$I \neq 0$	Q^*	SelI	EnI	SelR	EnR	Q	D	$D = (I \neq 0)$
0	0	1	1	1	1	0	0	$SelI = \overline{Q^*}$
0	1	0	1	0	1	0	0	$EnI = 1$
1	0	1	1	1	1	1	1	$SelR = \overline{Q^*}$
1	1	0	1	0	1	1	1	$EnR = 1$

□ Multiplicador por somas sucessivas

- ASM do módulo de controlo



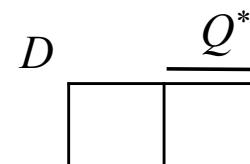
Implementação com *flip-flops* do tipo *D*



- Dois estados:
- 1 *flip-flop* tipo *D*
 - 1 mapa de *Karnaugh* de 1 variável

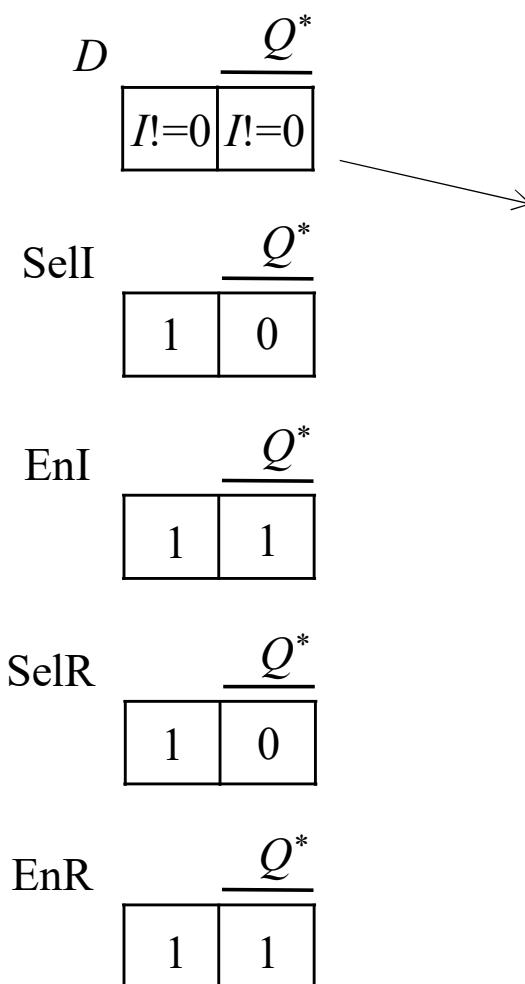
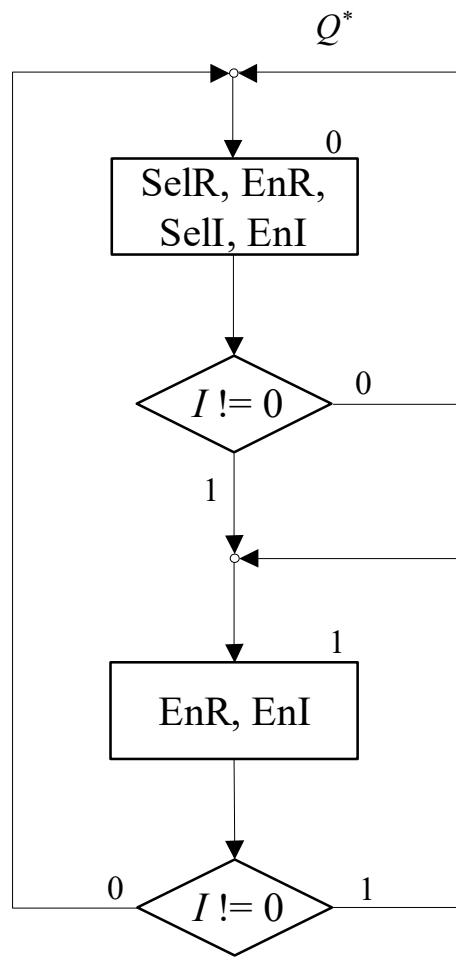
Tabela de transição de estados do *flip-flop D*:

Q^*	Q	D
0	0	0
0	1	1
1	0	0
1	1	1





□ Multiplicador por somas sucessivas



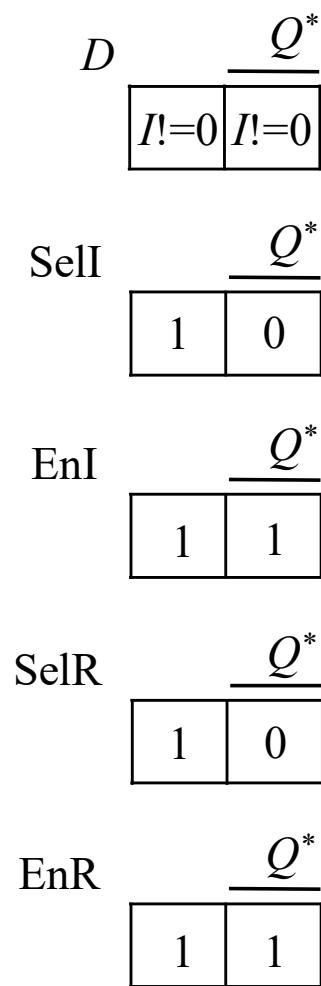
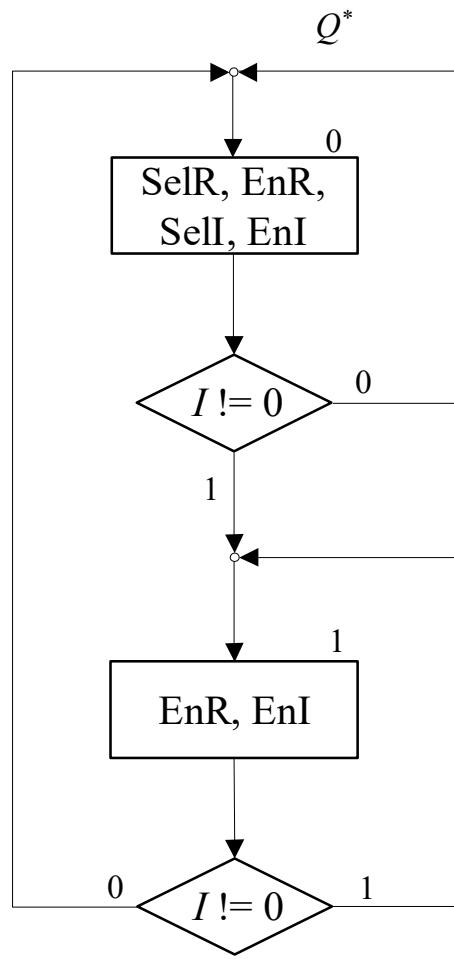
Preenchimento dos mapas de *Karnaugh*

Transita-se para o estado 1 sempre que $I!=0$.

A utilização de *flip-flops* do tipo *D* considera as transições para 1



□ Multiplicador por somas sucessivas



Extração de expressões simplificadas em mapas de Karnaugh com variáveis inseridas:

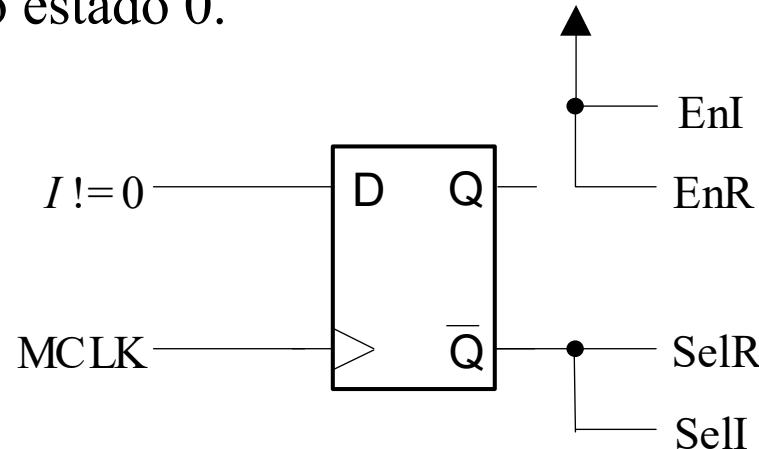
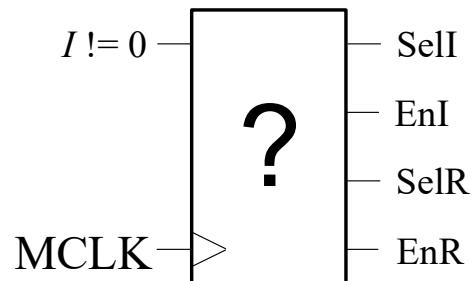
- Extrair os “1”s presentes no mapa, considerando como “0” as variáveis, e aproveitando os eventuais *don't care* para simplificação;
- Extrair as variáveis inseridas, considerando também os “1”s como *don't care*.

$$D = (I! = 0)$$

$$\begin{aligned}\text{SelI} &= \overline{Q^*} & \text{EnI} &= 1 \\ \text{SelR} &= \overline{Q^*} & \text{EnR} &= 1\end{aligned}$$

□ Multiplicador por somas sucessivas

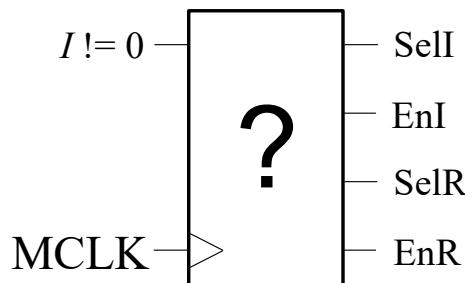
Assume-se que o arranque se dá no estado 0.



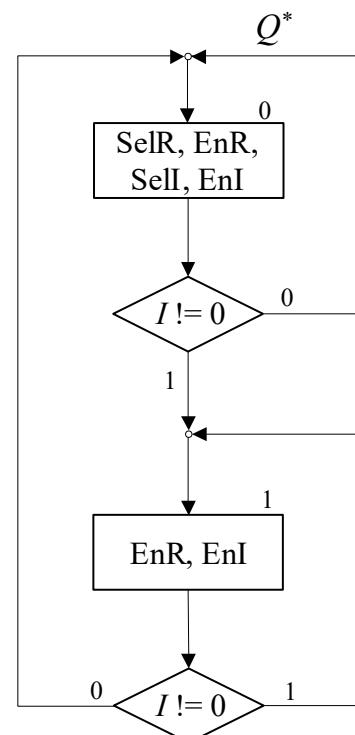
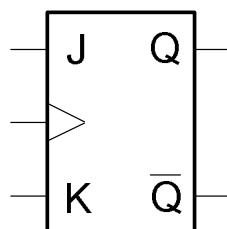
- I só fica a 0 no fim de tudo, ou seja, $(I \neq 0) == 0$. Até aí, $(I \neq 0) == 1$. $(I == 0)$ identifica o fim do cálculo e a apresentação do valor final do resultado em R .
- São os seletores dos *multiplexers* que determinam o encaminhamento dos dados.
- Este exemplo evidencia operações de cálculo (somas), aplicadas a dois operandos tal que o resultado é a multiplicação desses operandos, um pelo outro.

□ Multiplicador por somas sucessivas

- ASM do módulo de controlo



Implementação com *flip-flops* do tipo $J-K$

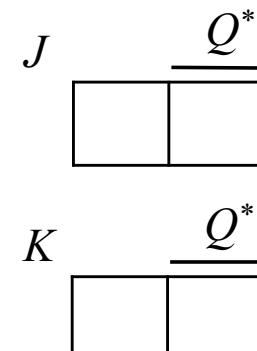


Dois estados:

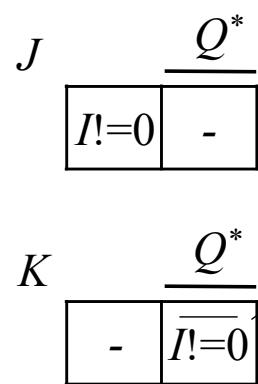
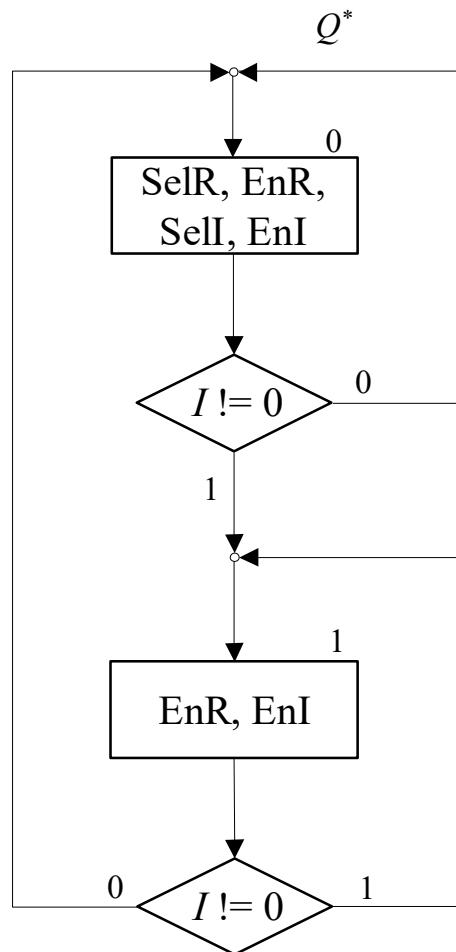
- 1 *flip-flop* tipo $J-K$
- 2 mapas de *Karnaugh* de 1 variável

Tabela de transição de estados do *flip-flop* $J-K$:

Q^*	Q	J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0



□ Multiplicador por somas sucessivas



Preenchimento dos mapas de *Karnaugh*

- Se $\overline{I!} = 0$, transita de 1 para 0
- Se $I! = 0$, transita de 1 para 1

$$J = (I! = 0)$$

$$K = \overline{(I! = 0)}$$

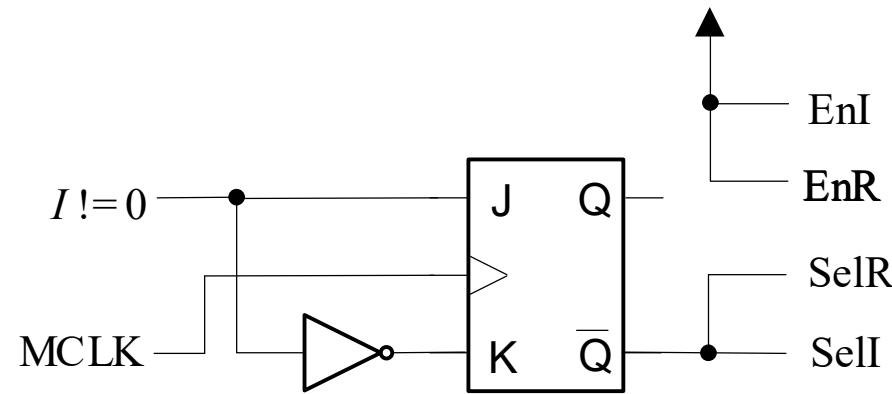
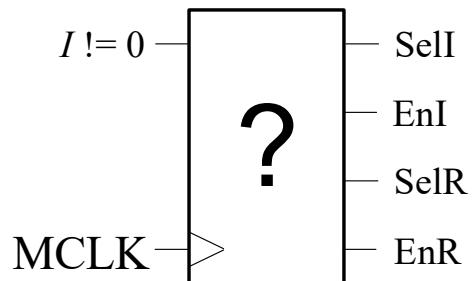
Q^*	Q	J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

No estado 0, as transições possíveis são $0 \rightarrow 0$ ou $0 \rightarrow 1$. Pela tabela de transição, J identifica-se com Q , por isso, vai-se pelo “1”.

No estado 1, as transições possíveis são $1 \rightarrow 0$ ou $1 \rightarrow 1$. Pela tabela de transição, K fica com a negação de Q , por isso, vai-se pelo “0”.



□ Multiplicador por somas sucessivas



As expressões lógicas das saídas mantêm-se, dado que não dependem do tipo de *flip-flop* usado na implementação.

$$\begin{aligned} J &= (I \neq 0) & K &= \overline{(I \neq 0)} \\ \text{SelI} &= \overline{Q^*} & \text{EnI} &= 1 \\ \text{SelR} &= \overline{Q^*} & \text{EnR} &= 1 \end{aligned}$$



□ Multiplicador por somas sucessivas

O tipo de registo usado na construção do módulo funcional é o registo *edge-triggered* com *enable*. Este é composto internamente por um *multiplexer* e um registo *edge-triggered*. Logo, uma implementação do registo com *enable* no Arduino, respondendo literalmente à definição dada anteriormente, poderá ser:

```
byte MUX_2x1(boolean S, byte IN_0, byte IN_1) {  
    return S ? IN_1 : IN_0;  
}  
  
byte registoComEnable(boolean enable, byte D, byte Q) {  
    return MUX_2x1(enable, Q, D);  
}
```



□ Arquitetura Harvard

- A arquitetura Harvard é composta pelos seguintes elementos:
 - **Memória de código:** onde está o programa a cumprir. Esta memória é só de leitura. Para se cumprir um outro programa, diferente do atual em execução, a memória deverá ser regravada.
 - **Memória de dados:** onde estão as variáveis manipuladas pelo programa (operandos e resultados).
 - **CPU (*Central Processing Unit*):** onde o programa é cumprido de forma sequencial, ou seja, onde o algoritmo é executado.
- A arquitetura Harvard é uma arquitetura RISC (*Reduced Instruction Set Computer*), em oposição à CISC (*Complex Instruction Set Computer*).



☐ Elementos básicos de um CPU

- O CPU é especificado por alguns elementos constituintes básicos, um conjunto de instruções (*instruction set*) e um outro conjunto de elementos que permitem o cumprimento das instruções estabelecidas.
- Elementos básicos de um CPU:
 - Registo A (acumulador)
 - ALU (*Arithmetic and Logic Unit*) - unidade lógica e aritmética
 - Módulo de controlo
 - *Flags* (conjunto de *flip-flops* com informação específica)
 - Registo PC (*Program Counter*), aka IP (Instruction Pointer)



□ Formato das instruções do CPU

- O *instruction set* determina a gama de algoritmos que é possível realizar e a estrutura interna do módulo funcional (onde se situam os elementos básicos)
- As instruções podem ser constituídas por:
 - Um código apenas
 - Um código e parâmetros
- Quantos mais bits se tiver para codificar as instruções, mais estruturada será a codificação e mais simples o descodificador do módulo de controlo, porém, maior será o *data bus* (DB) da memória de código, sendo desvantajoso



□ Formato das instruções do CPU

- Codificação das instruções:
 - As instruções devem ser o mais curtas possível, ou seja, codificadas usando o menor número possível de bits. O comprimento dos parâmetros que hajam em certas instruções condiciona este critério
 - Os códigos atribuídos às instruções devem ser únicos, ou seja, não podem haver instruções com o mesmo código, contemplando também a presença de possíveis parâmetros
 - Os bits com valor constante, isto é, que não façam parte de parâmetros, deverão servir para fazer a distinção entre instruções, ou até entre classes de instruções, se possível



□ Formato das instruções do CPU

- Classes de instruções:
 - Transferência de informação
 - Transferência de valores (leitura e escrita) entre registo internos do CPU, ou entre estes registo e a memória de dados
 - Aritméticas e/ou lógicas
 - Cálculo aritmético ou lógico, realizado na ALU (módulo combinatório interno do CPU)
 - Controlo de fluxo
 - Alteração à sequência de execução do programa, de forma incondicional ou condicional (*branching* - em função das *flags*)



□ Formato das instruções do CPU

- Conjunto de instruções (*instruction set*):

Classes de instruções	Instrução	Funcionalidade
Transferência de informação	MOV A, @Rn	A = (Rn)
	MOV @Rn, A	@Rn = A
	MOV A, Rn	A = Rn
	MOV Rn, A	Rn = A
Aritméticas	ADDC A, Rn	A = A + Rn + Cy
	SUBB A, Rn	A = A - Rn - Cy
Controlo de fluxo	JNC rel6	Se (!Cy) PC += rel6
	JNZ rel6	Se (!Z) PC += rel6
	JMP rel6	PC += rel6



□ Formato das instruções do CPU

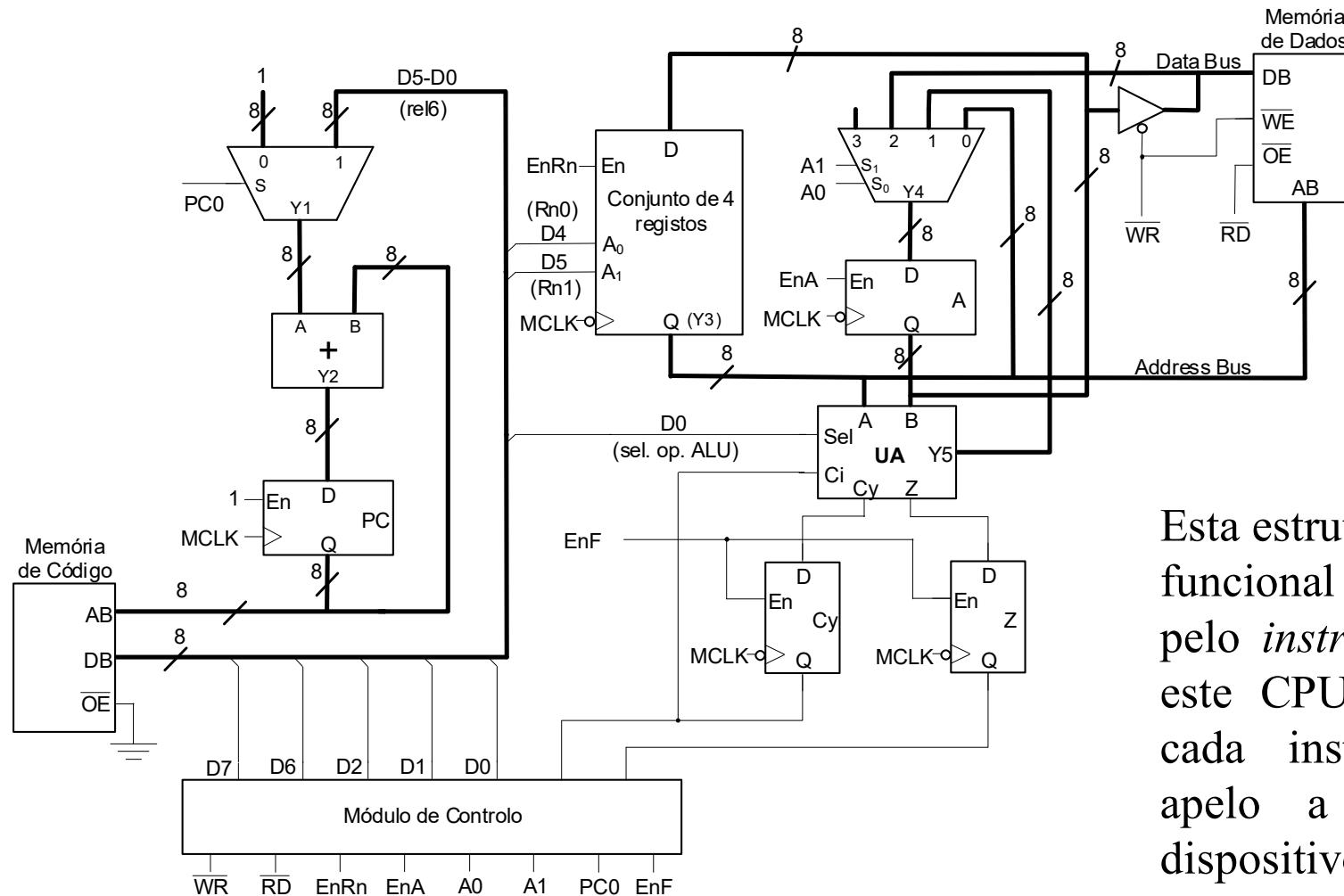
- Conjunto de instruções (*instruction set*):

Instrução	Parâmetros	Codificação									
		D7	D6	D5	D4	D3	D2	D1	D0	Hexa.	
MOV A, @Rn	Rn	1	1	Rn1	Rn0	0	0	0	0	0x-0	
MOV @Rn, A	Rn	1	1	Rn1	Rn0	0	0	0	1	0x-1	
MOV A, Rn	Rn	1	1	Rn1	Rn0	0	0	1	0	0x-2	
MOV Rn, A	Rn	1	1	Rn1	Rn0	0	0	1	1	0x-3	
ADDC A, Rn	Rn	1	1	Rn1	Rn0	0	1	0	0	0x-4	
SUBB A, Rn	Rn	1	1	Rn1	Rn0	0	1	0	1	0x-5	
JNC rel6	rel6	1	0	r5	r4	r3	r2	r1	r0	0x--	
JNZ rel6	rel6	0	1	r5	r4	r3	r2	r1	r0	0x--	
JMP rel6	rel6	0	0	r5	r4	r3	r2	r1	r0	0x--	

- A distinção entre instruções é feita com o valor de 5 ($D_{7, 6, 2, 1, 0}$) dos 8 bits
- Os parâmetros vão embedidos no código da instrução



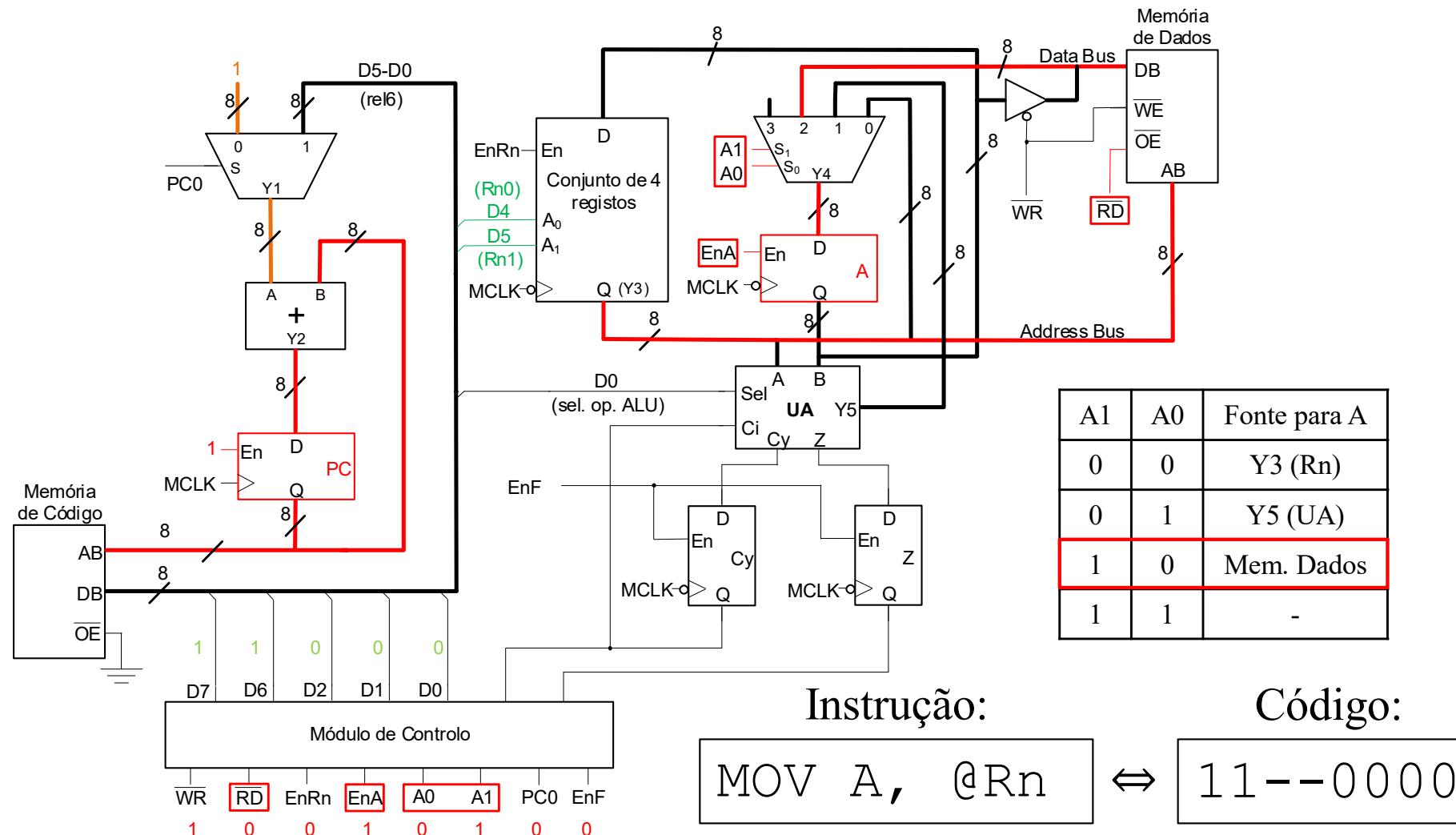
□ Módulo funcional do CPU



Esta estrutura de módulo funcional é determinada pelo *instruction set* que este CPU executa. Em cada instrução, faz-se apelo a determinados dispositivos da estrutura.

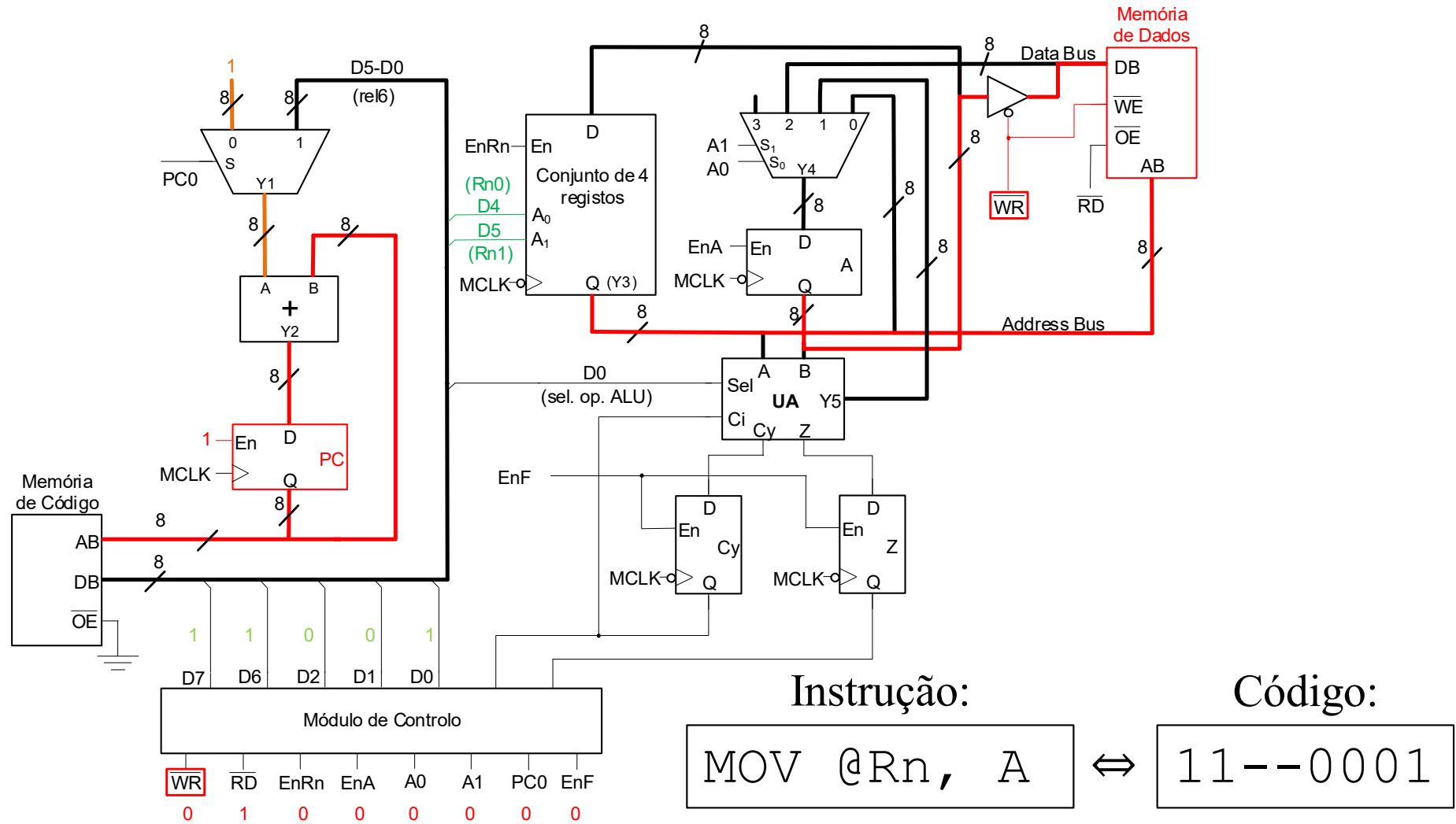


□ Módulo funcional do CPU

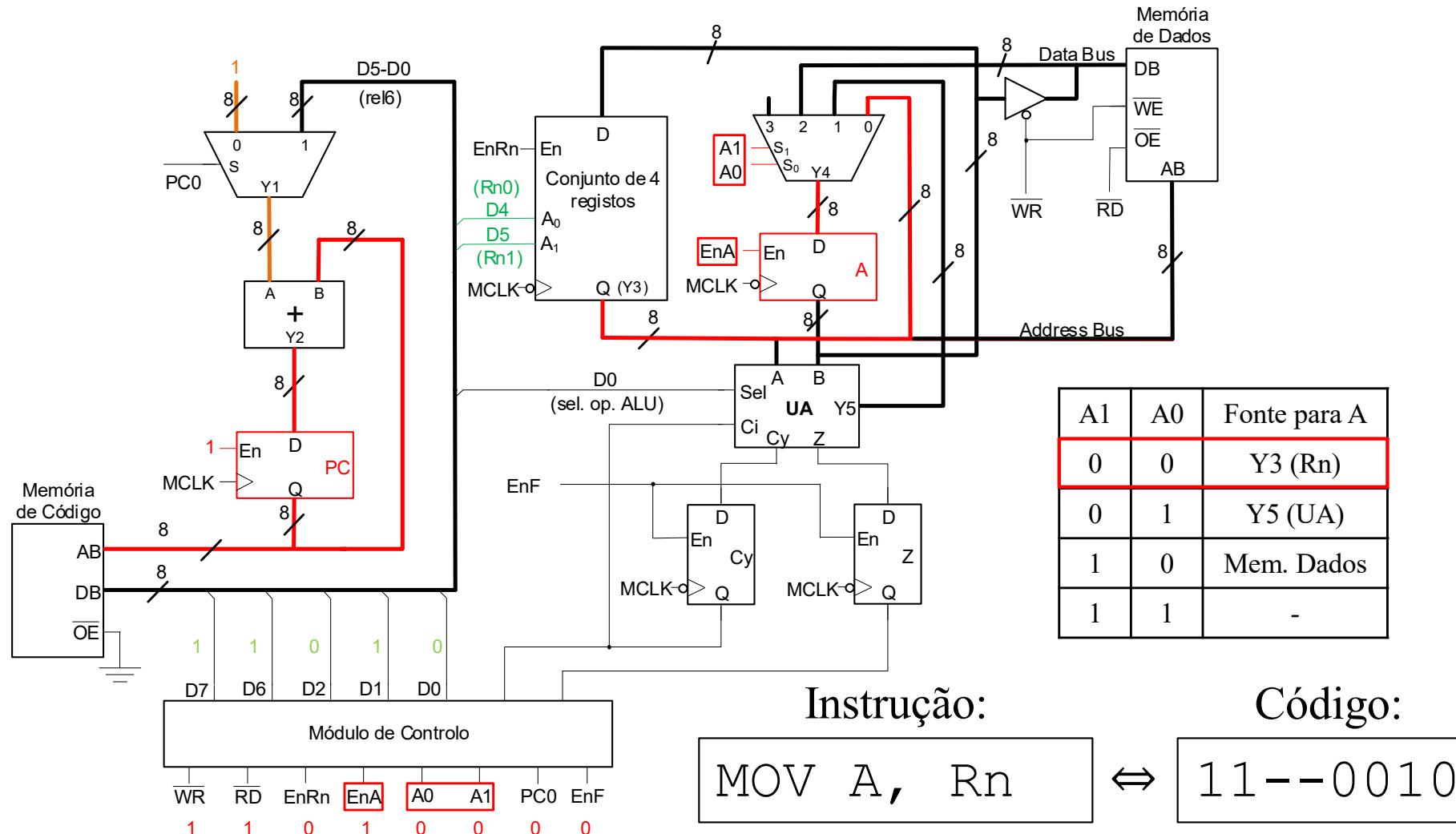




□ Módulo funcional do CPU

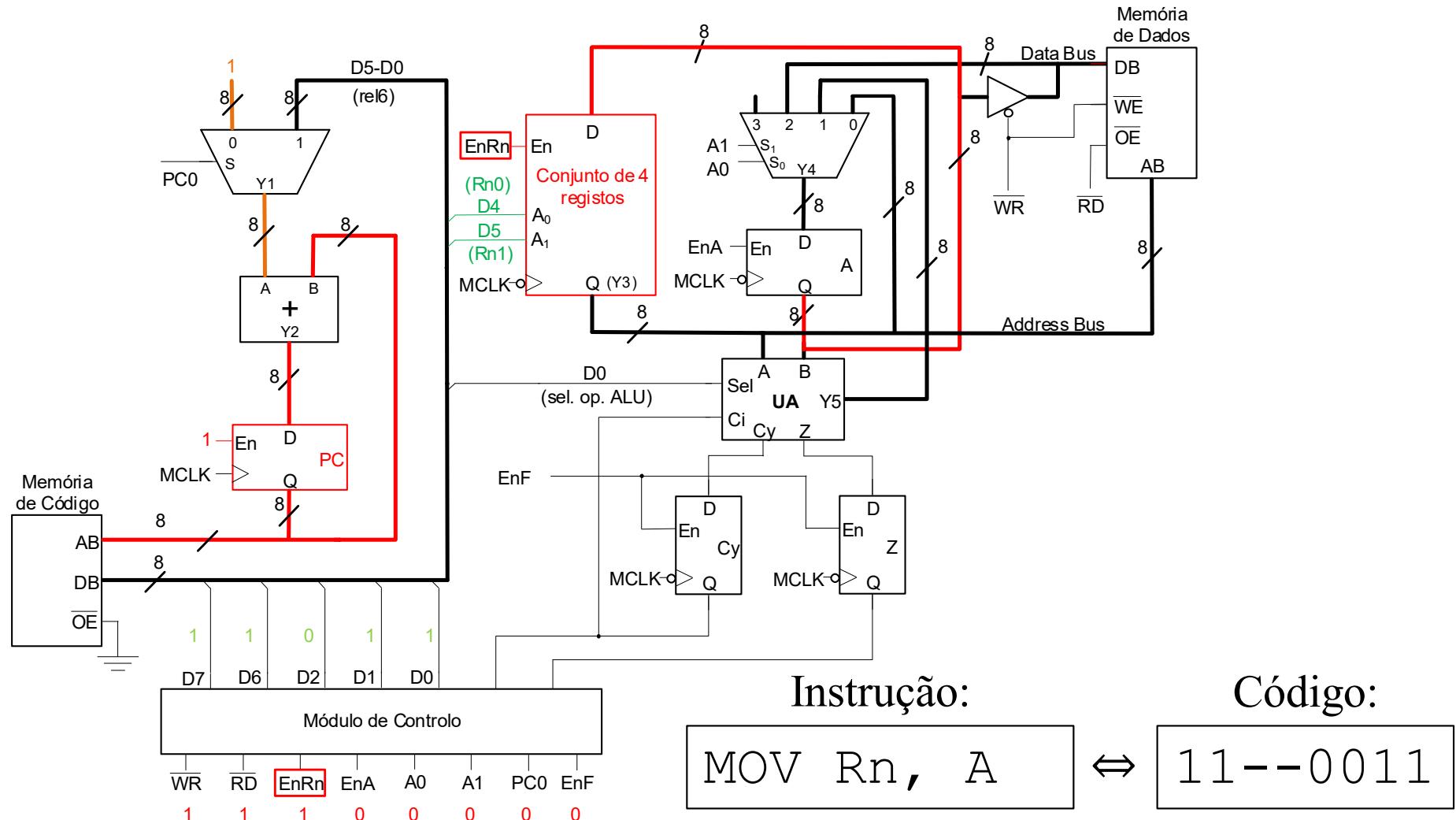


□ Módulo funcional do CPU



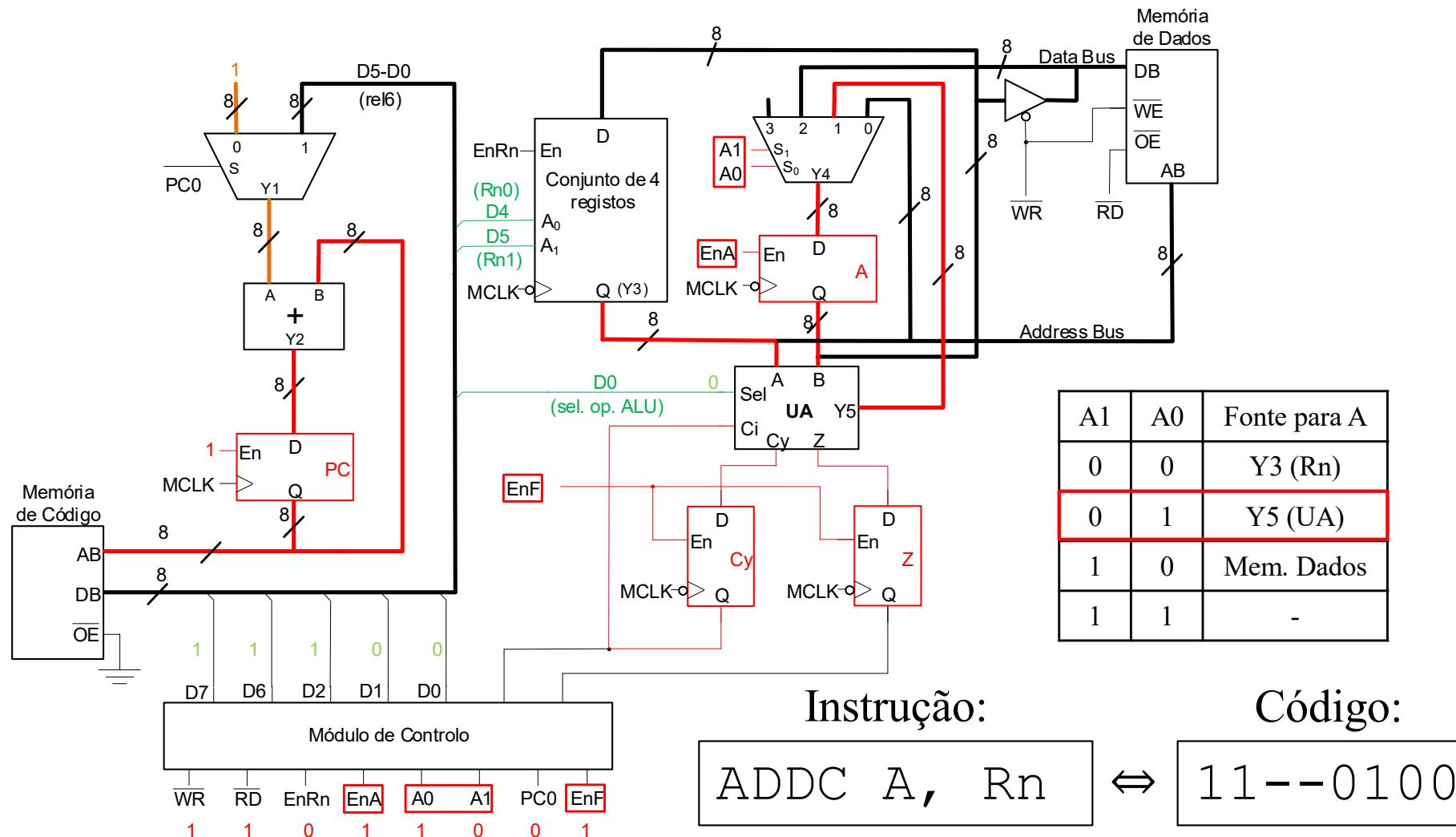


□ Módulo funcional do CPU



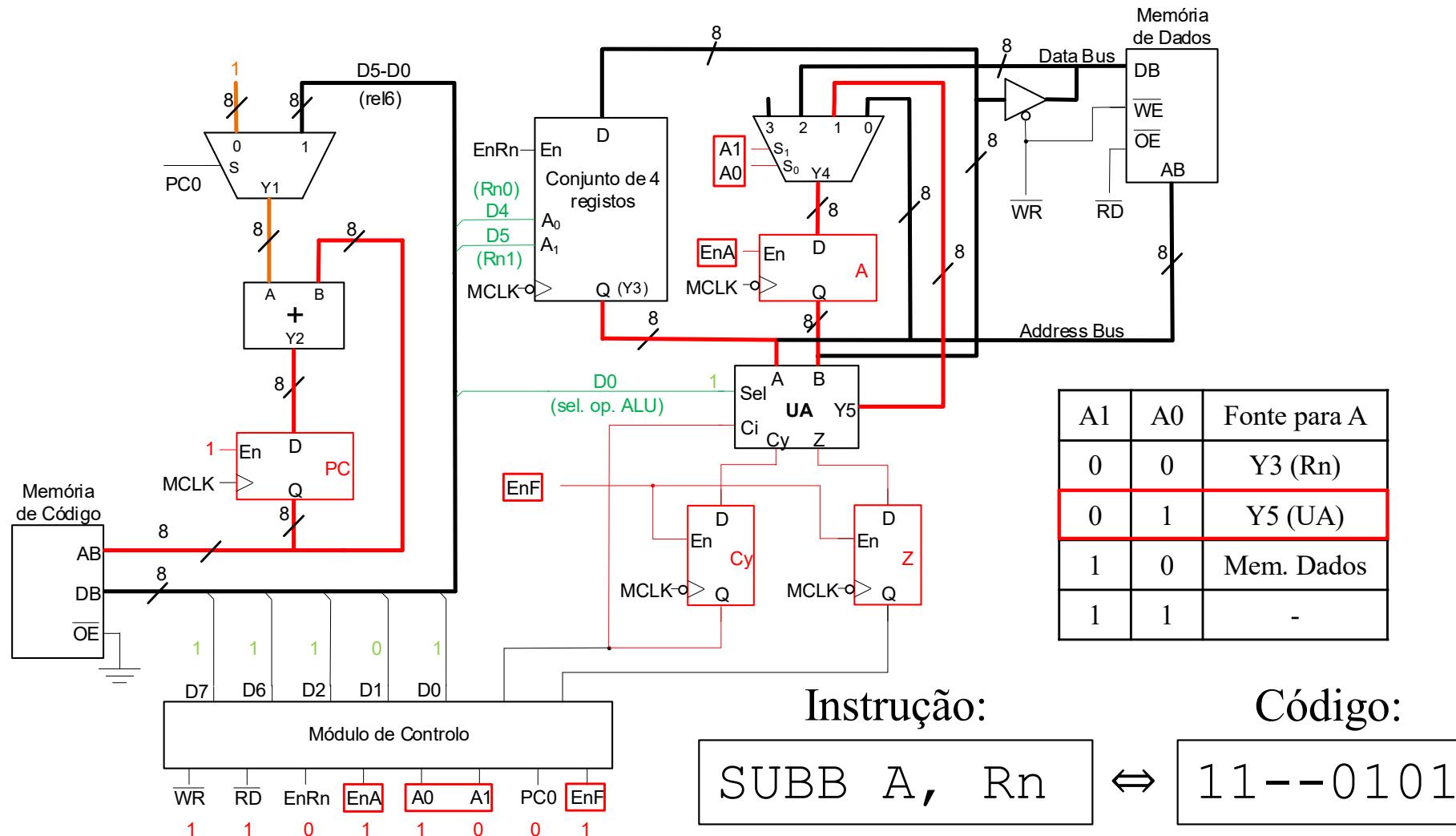


□ Módulo funcional do CPU



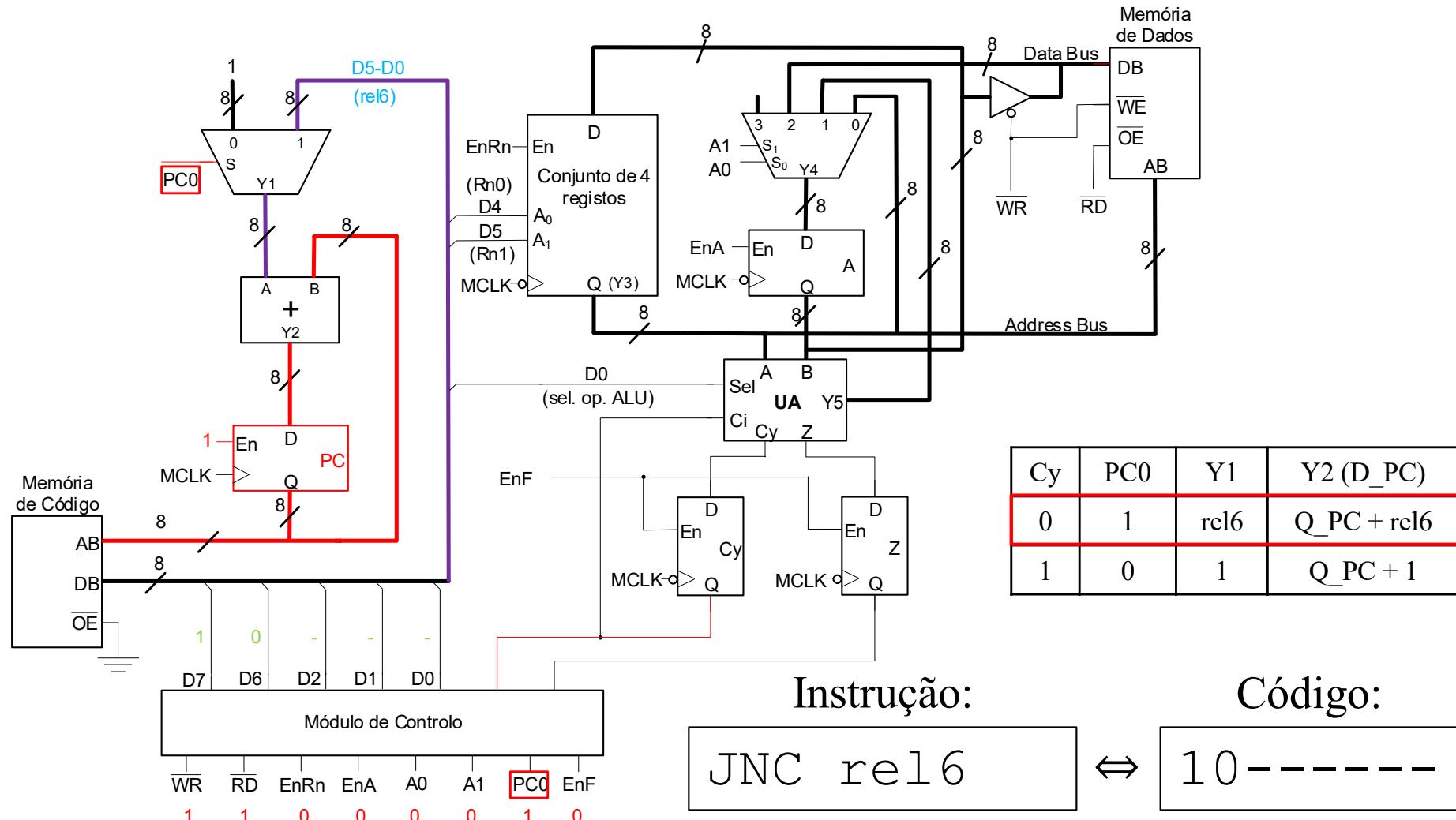


□ Módulo funcional do CPU



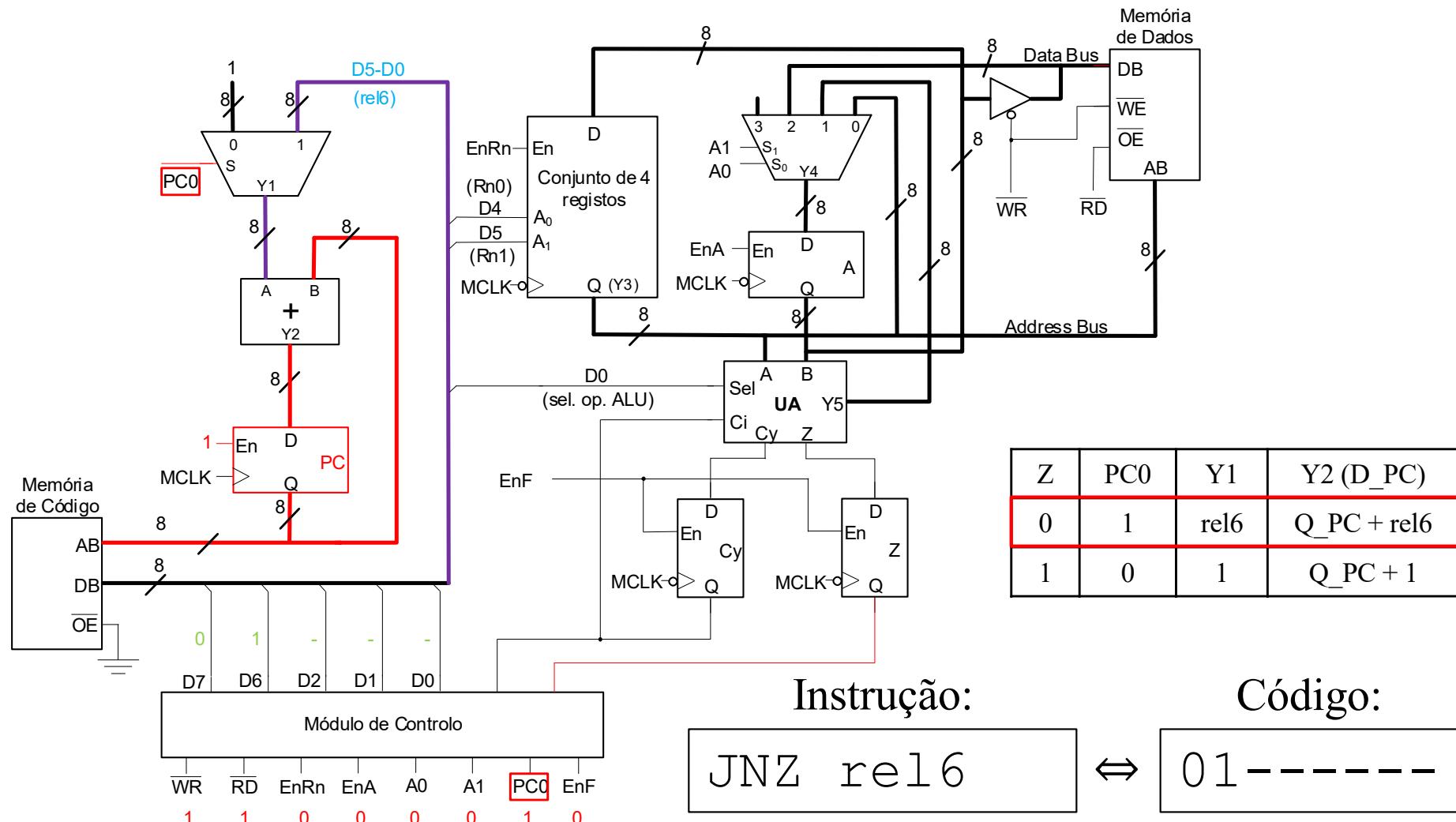


□ Módulo funcional do CPU



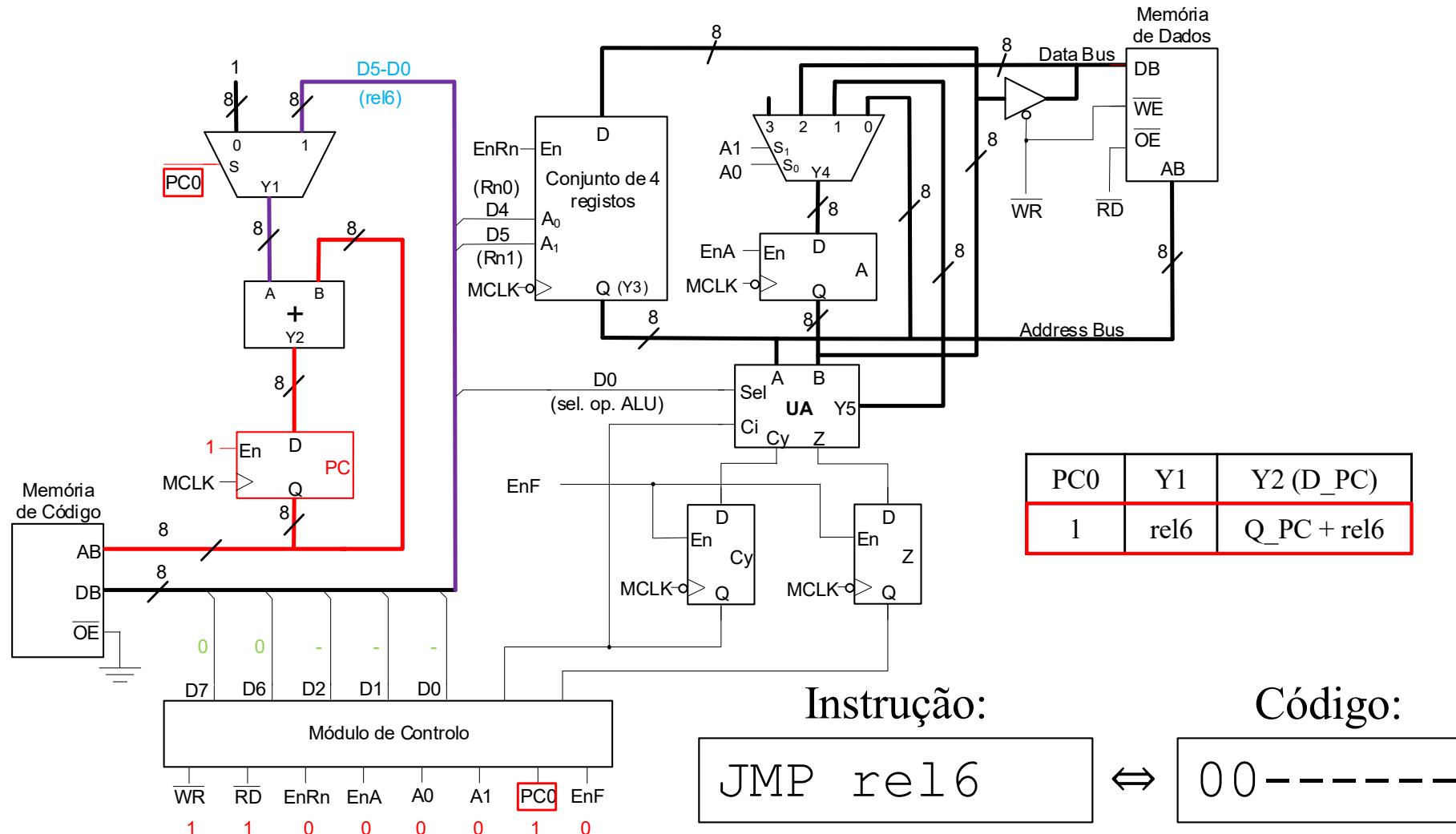


□ Módulo funcional do CPU





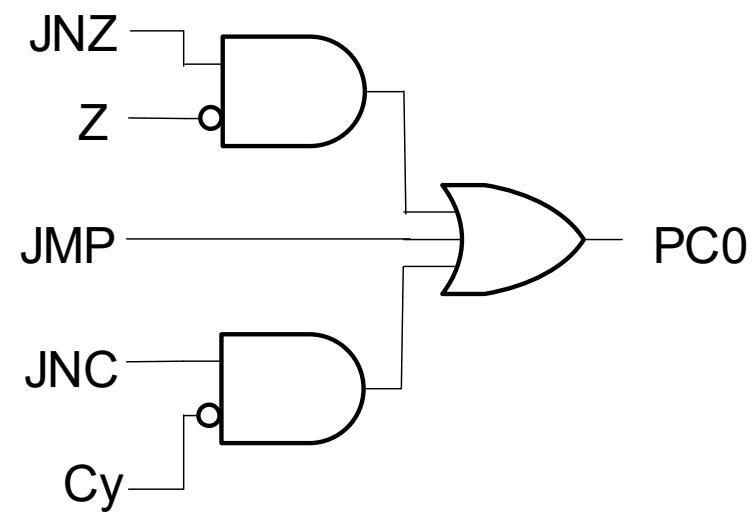
□ Módulo funcional do CPU



☐ Redução da dimensão da ROM do módulo de controlo

É possível reduzir a dimensão da ROM do módulo de controlo, recorrendo à remoção das entradas provenientes da *flag Carry* (Cy) e da *flag Zero* (Z). Por cada entrada a menos no módulo de controlo, a sua dimensão decresce para metade. Assim, com menos duas entradas, este passa para $\frac{1}{4}$ das posições originais (de 128 para 32 posições).

Contudo, o MUX com saída Y1 continua a depender do sinal PC0, mas agora este é fornecido, não pelo módulo de controlo, mas por uma malha combinatória auxiliar.





□ Exemplos de programas

Vão ser apresentados alguns exemplos de programas simples que podem ser realizados sobre este CPU, respeitantes a pequenos troços de código.

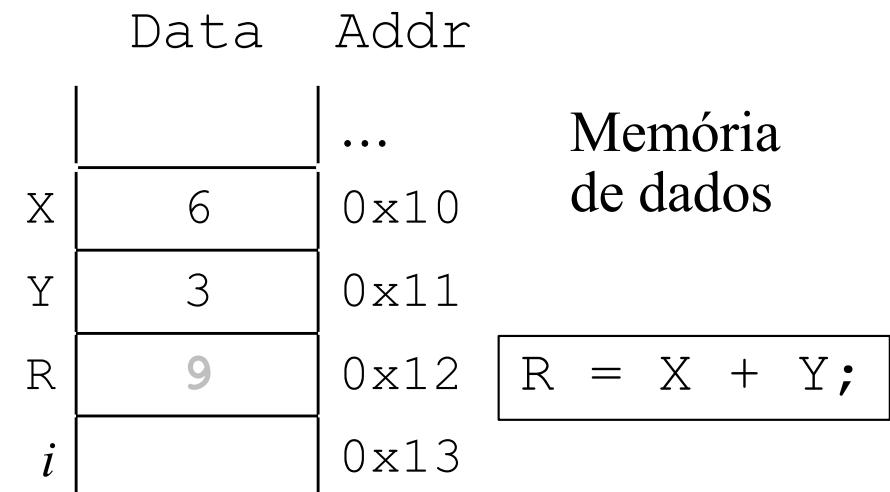
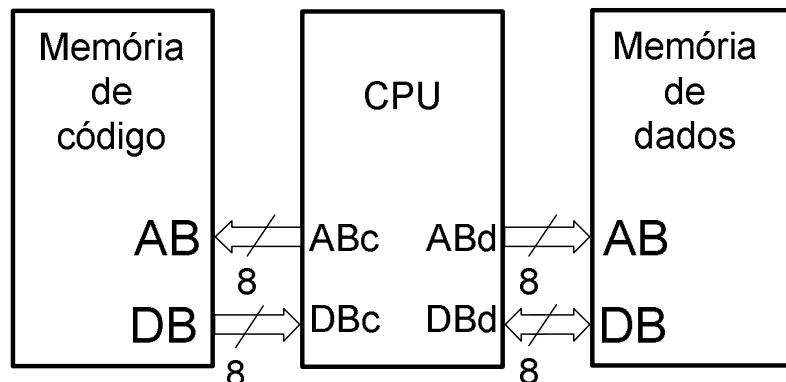
Admite-se que R0 contém já o endereço de X ($0x10$), que R1 contém o endereço de Y ($0x11$) e que R2 contém o endereço de R ($0x12$).

Programas a realizar:

- $R = X + Y;$
- $R = -X;$
- $X *= 3;$
- $R = X > Y ? X : Y;$
- $R = X > Y ? X : X == Y ? 0 : Y;$



□ Exemplos de programas



Programa (linguagem assembly)

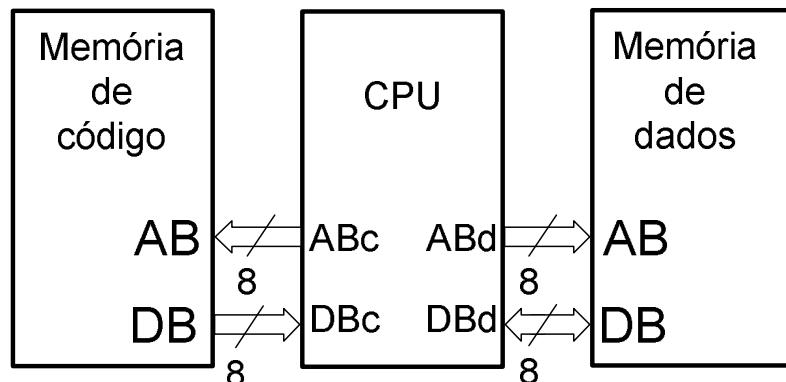
PC	Address	Assembly
0x00		MOV A, @R0 ; A = X
0x01		MOV R3, A ; R3 = X
0x02		MOV A, @R1 ; A = Y
0x03		ADDC A, R3 ; A = X + Y
0x04		MOV @R2, A ; R = X + Y
0x05		JMP 0 ; HALT

Código (linguagem máquina)

Address	Data	Memory
0x00	11000000	0xC0
0x01	11110011	0xF3
0x02	11010000	0xD0
0x03	11110100	0xF4
0x04	11100001	0xE1
0x05	00000000	0x00



□ Exemplos de programas



	Data	Addr	Memória de dados
X	+6 (0x06)	0x10	...
R	-6 (0xFA)	0x11	
		0x12	
		0x13	$R = -X;$

Programa (linguagem *assembly*)

PC

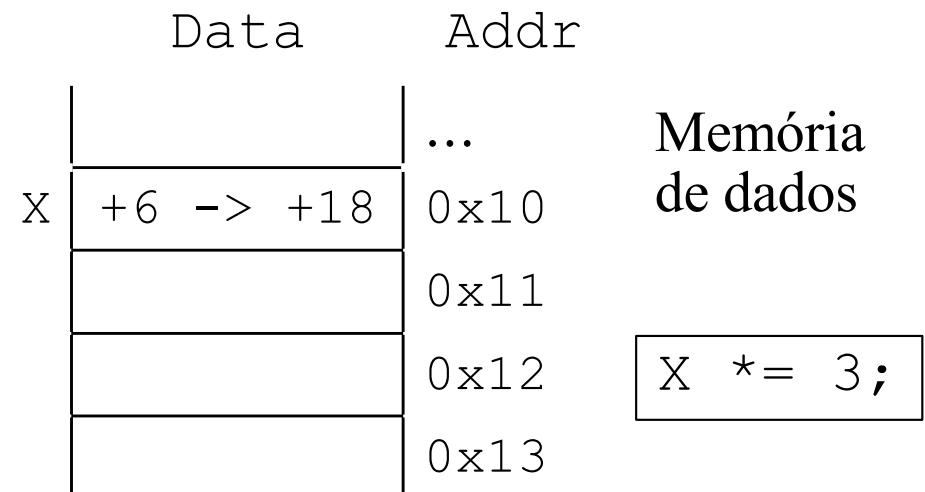
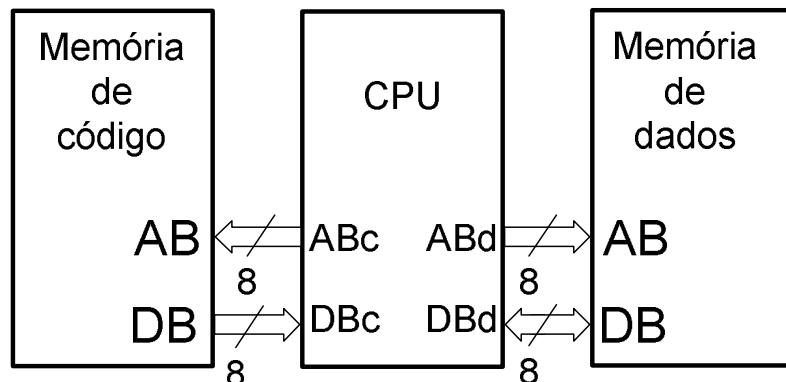
0x00	MOV	A, @R0 ; A = X
0x01	MOV	R3, A ; R3 = X
0x02	SUBB	A, R3 ; A = 0
0x03	SUBB	A, R3 ; A = -X
0x04	MOV	@R1, A ; R = -X
0x05	JMP	0 ; HALT

Código (linguagem máquina)

11000000	0xC0	Memória de código
11110011	0xF3	
11110101	0xF5	
11110101	0xF5	
11010001	0xD1	
00000000	0x00	



□ Exemplos de programas



Programa (linguagem *assembly*)

PC

0x00	MOV	A, @R0 ; A = X
0x01	MOV	R3, A ; R3 = X
0x02	ADDC	A, R3 ; A = 2 * X
0x03	ADDC	A, R3 ; A = 3 * X
0x04	MOV	@R0, A ; X = 3 * X
0x05	JMP	0 ; HALT

Código (linguagem máquina)

11000000	0xC0	
11110011	0xF3	
11110100	0xF4	
11110100	0xF4	
11000001	0xC1	
00000000	0x00	

Memória de código



□ Exemplos de programas

Deixar em R o maior dos números, entre X e Y . Se forem iguais, fica $R = Y$.

PC		R = X > Y ? X : Y;
0x00	MOV A, @R0 ; A = X	11 00 0000 0xC0
0x01	MOV R3, A ; R3 = X	11 11 0011 0xF3
0x02	MOV A, @R1 ; A = Y	11 01 0000 0xD0
0x03	SUBB A, R3 ; A = Y - X	11 11 0101 0xF5
0x04	JNC +4 ; Y ≥ X	10 000100 0x84
0x05	MOV A, R3 ; X > Y	11 11 0010 0xF2
0x06	MOV @R2, A ; R = X	11 10 0001 0xE1
0x07	JMP 0 ; HALT	00 000000 0x00
0x08	MOV A, @R1 ; A = Y	11 01 0000 0xD0
0x09	MOV @R2, A ; R = Y	11 10 0001 0xE1
0x0A	JMP -3 ; salto p/ HALT	00 111101 0x3D



□ Exemplos de programas

Deixar em R o maior dos números, entre X e Y . Se forem iguais, fica $R = 0x00$.

PC		R = X > Y ? X : X == Y ? 0 : Y;
0x00	MOV A, @R0 ; A = X	11000000 0xC0
0x01	MOV R3, A ; R3 = X	11110011 0xF3
0x02	MOV A, @R1 ; A = Y	11010000 0xD0
0x03	SUBB A, R3 ; A = Y - X	11110101 0xF5
0x04	JNC +4 ; Y ≥ X	10000100 0x84
0x05	MOV A, R3 ; X > Y → A = X	11110010 0xF2
0x06	MOV @R2, A ; R = X	11100001 0xE1
0x07	JMP 0 ; HALT	00000000 0x00
0x08	JNZ +3 ; Y > X	01000011 0x43
0x09	MOV @R2, A ; Y == X → R = 0	11100001 0xE1
0x0A	JMP -3 ; salto p/ HALT	00111101 0x3D
0x0B	MOV A, @R1 ; A = Y	11010000 0xD0
0x0C	MOV @R2, A ; R = Y	11100001 0xE1
0x0D	JMP -6 ; salto p/ HALT	00111010 0x3A



Sensores e Atuadores com interfaces digitais



□ Sensores e Atuadores com interfaces digitais

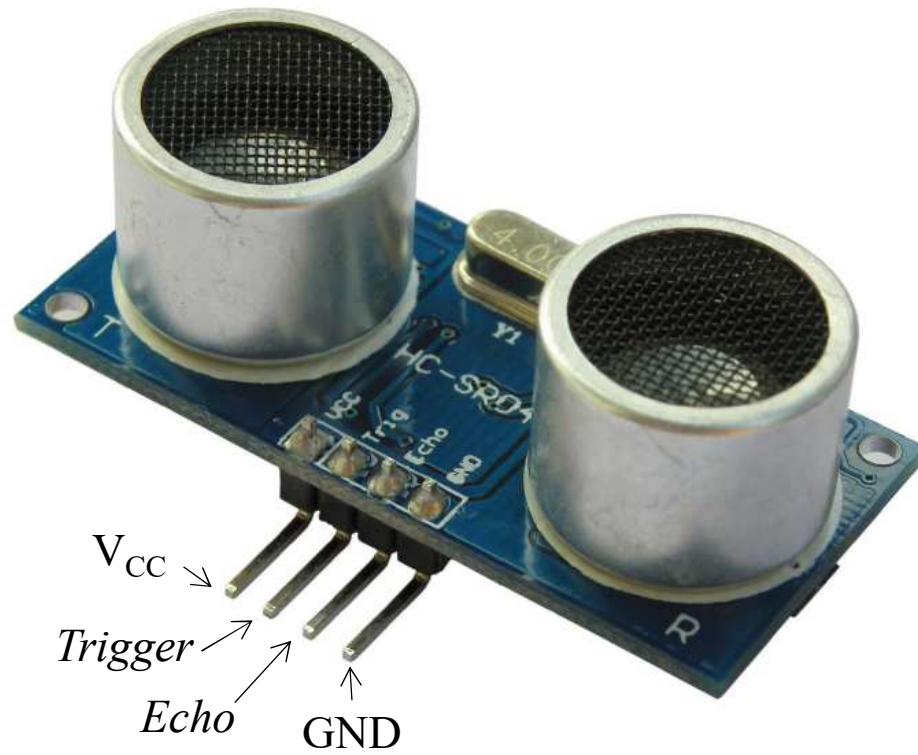
Irá dar-se conhecimento de alguns dispositivos sensores e atuadores com interfaces digitais, alguns deles fazendo uso da interface I²C. A saber:

1. Sensor de distância por ultrassons HC-SR04;
2. Sensores e atuadores com interface I²C:
 - a) Display alfanumérico;
 - b) Giroscópio LSM303D;
 - c) Acelerómetro e magnetómetro L3GD20;
 - d) Sensor de temperatura e de pressão atmosférica BMP180



☐ Pinos de interface física

Pinos de interface física:



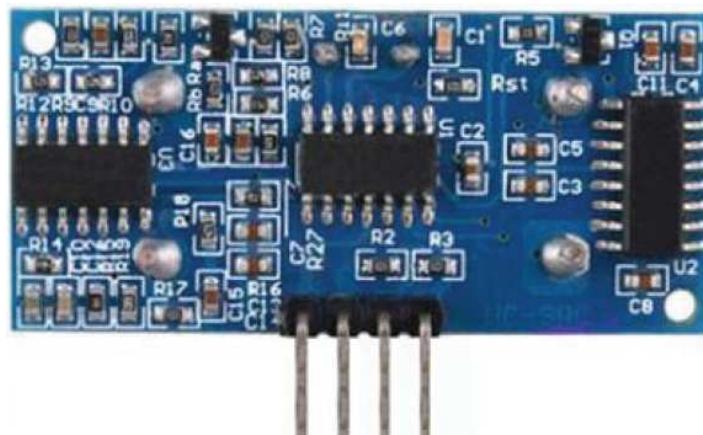
Pinos de interface física:

- V_{CC} – **Alimentação** (5 V).
- *Trigger* – **Entrada** de sinal pulsado que dá início a uma medição de distância.
- *Echo* – **Saída** de sinal pulsado, cujo intervalo de tempo a “1” é proporcional à distância medida.
- GND – **Referência** elétrica (massa).



☐ Aspetto físico do dispositivo e diagrama de radiação

Aspetto físico do dispositivo:



De trás



De frente

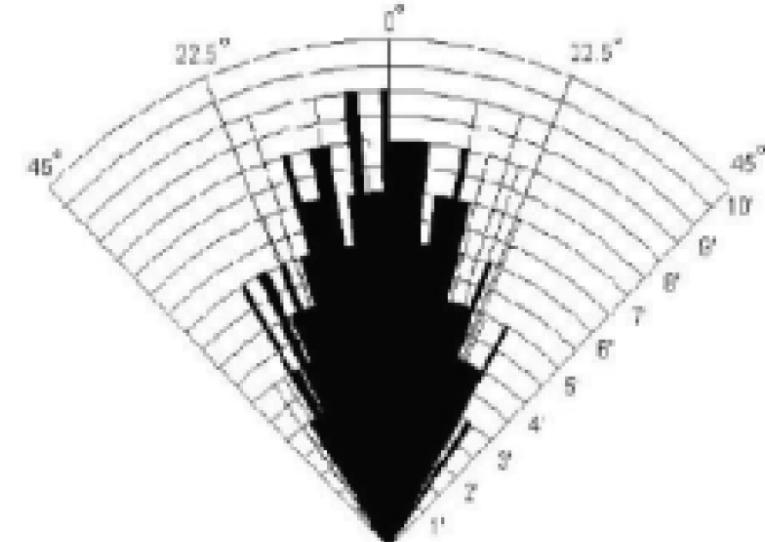


Diagrama de radiação ultrassónica

A energia emitida forma um cone no espaço e é através do tempo de voo da energia emitida e refletida que se consegue estimar a distância ao objeto.

Gama de distâncias: entre 2 cm e 4 m.



□ Processo de medição

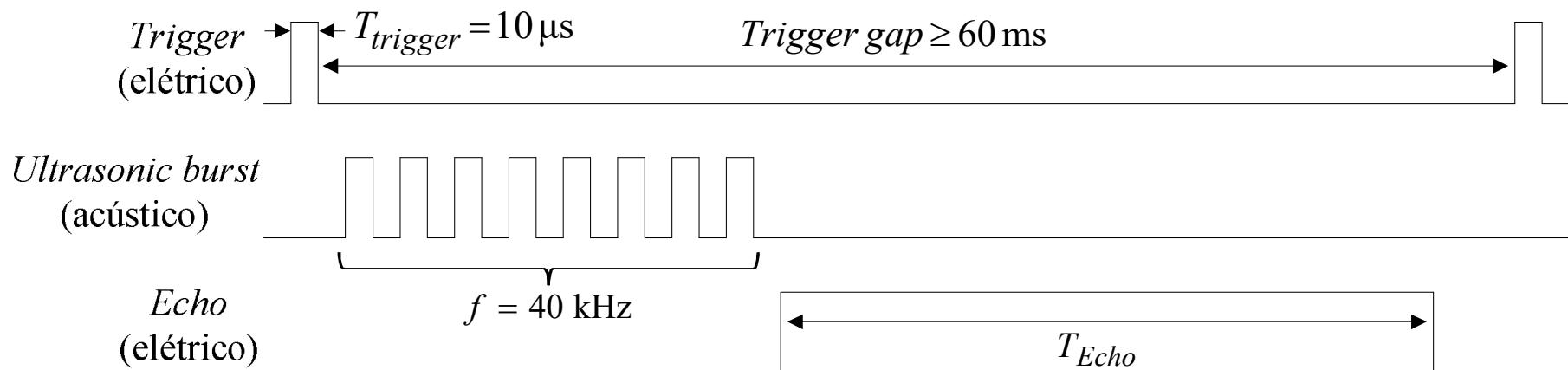
A medição da distância assenta na medição do tempo de ida e volta de um trem de impulsos que é gerado no momento do início da medição. O processo consiste dos seguintes passos:

1. Gerar um impulso de *Trigger* com duração de 10 µs (no mínimo);
2. De seguida, o dispositivo gera um trem de oito impulsos com uma frequência de 40kHz (ultrassónico);
3. Após o envio do trem de impulsos, o dispositivo coloca a sua saída de *Echo* a “1”, marcando o início da contagem do tempo de eco;
4. Quando o eco é recebido, a saída de *Echo* desce para 0;
5. Através da duração do impulso de *Echo*, estima-se a distância.



□ Processo de medição

Diagrama temporal:



$$343 [\text{m/s}] = \frac{2 \times dist [\text{cm}]}{T_{Echo} [\mu\text{s}]} \Leftrightarrow \frac{343 \times 100}{10^6} \left[\frac{\text{cm}}{\mu\text{s}} \right] = \frac{2 \times dist [\text{cm}]}{T_{Echo} [\mu\text{s}]} \Leftrightarrow \\ \Leftrightarrow dist [\text{cm}] = 0.01715 \times T_{Echo} [\mu\text{s}]$$

$$dist[\text{cm}] \cong \frac{T_{Echo} [\mu\text{s}]}{58}$$



❑ Processo de medição

- De acordo com o fabricante, se não forem detetados obstáculos à frente do feixe acústico, o tempo de voo nesta situação é de 38 ms.
- Se se verificar que a duração de T_{ECO} é superior a este valor, tal é indicativo de que não haverá eco de retorno.
- Por este motivo, o fabricante aconselha que o intervalo mínimo entre dois impulsos de *Trigger* (ou seja, entre duas medições consecutivas), seja de 60 ms.
- Certos objetos ou superfícies, pelo facto de exibirem características não reflectoras, irão introduzir erros ou dificuldades no processo de medição.



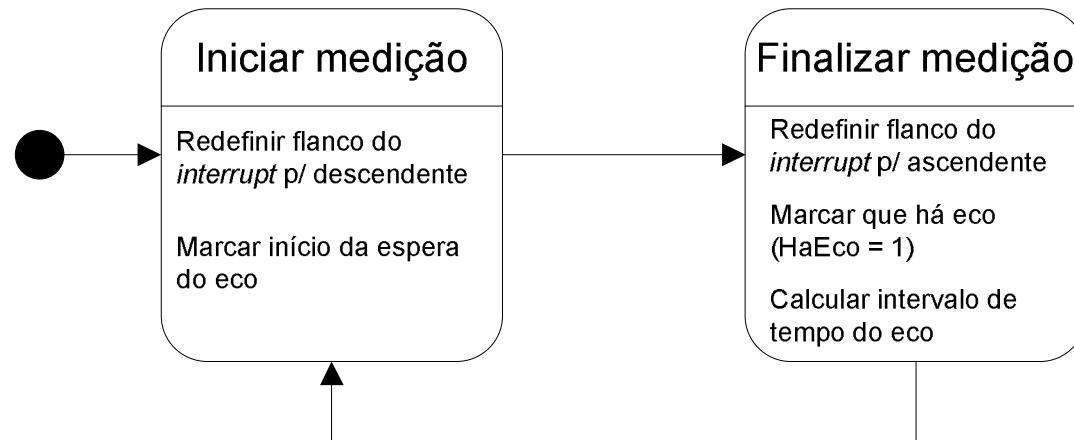
❑ Processo de medição

- A gestão do processo de medição pode ser feito recorrendo a autómatos para esse efeito
- Irão haver dois autómatos: um para lidar diretamente com o sensor de distância e outro para tratar a informação que o primeiro fornece, gerindo também a cadência das medições, bem como as condições e as temporizações inerentes às mesmas
- O autómato que lida diretamente com o sensor de distância funciona ativado por um *interrupt*, gerado em cada um dos flancos do sinal de *Echo*

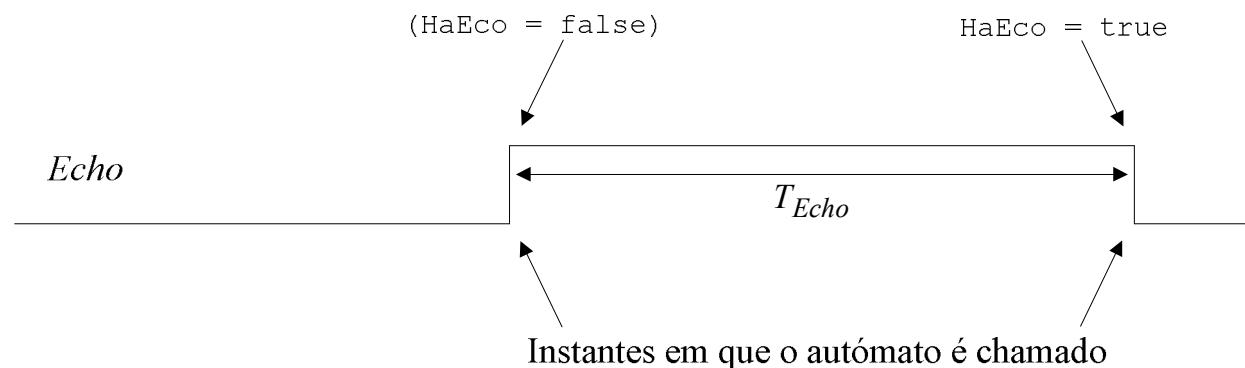


❑ Processo de medição

- Autómato para tratamento do sensor de distância



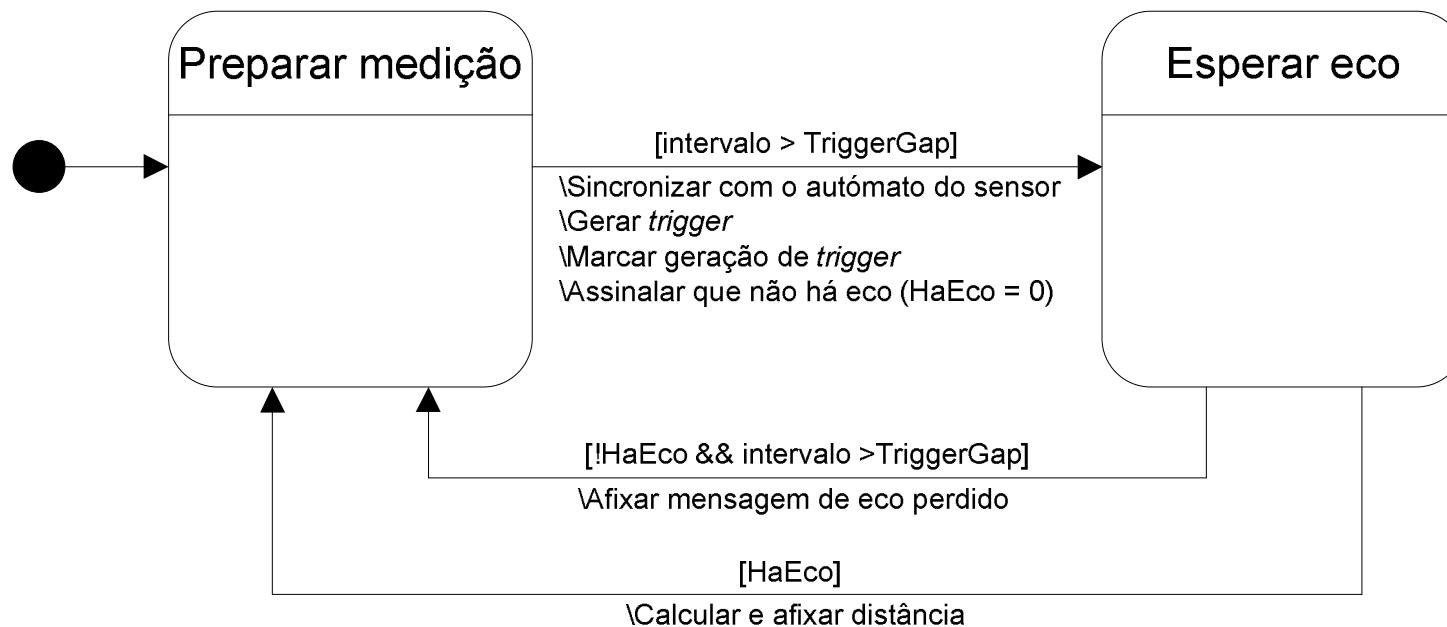
O autómato é ativado por *interrupt*, através do sinal de *Echo* do sensor





❑ Processo de medição

- Autómato para gestão da medição



Os autómatos comunicam através das variáveis globais HaEco e TEco.



□ Introdução

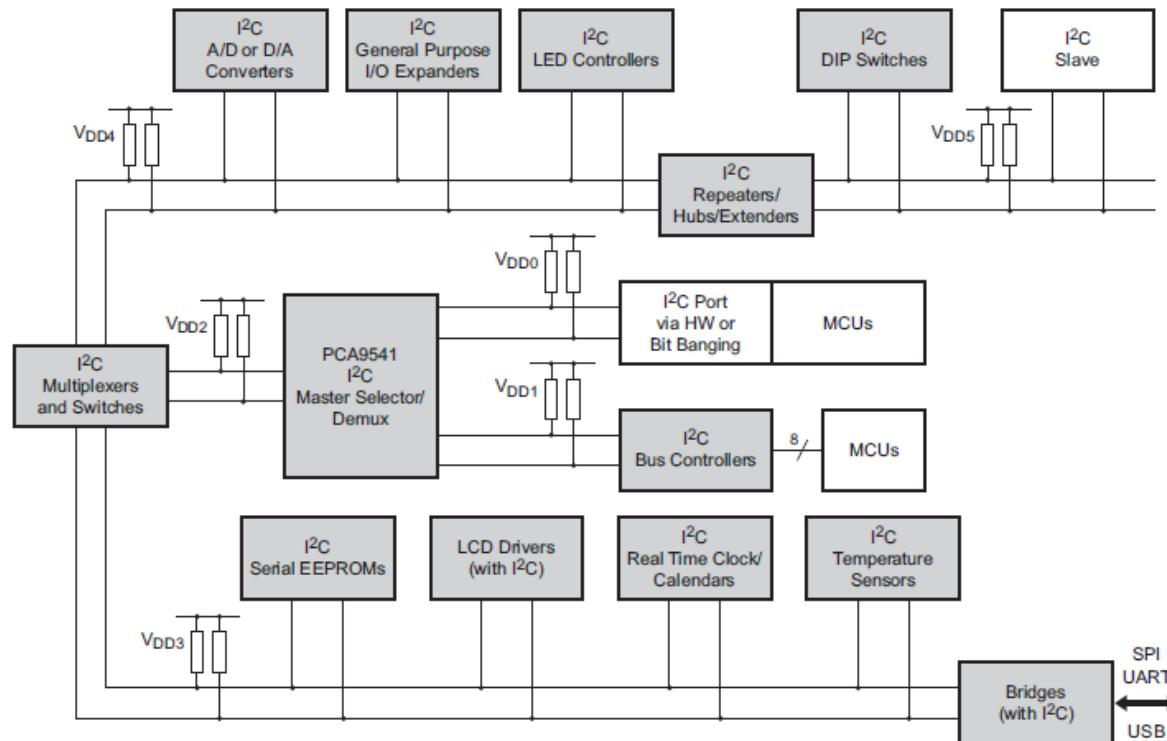
- I²C – *Inter-Integrated Circuit* (ou TWI – *Two Wire Interface*)
- Os dispositivos compatíveis com este protocolo incorporam uma interface no seu *chip* que lhes permite comunicar diretamente com outros dispositivos presentes no *bus* I²C.
- Dois sinais fundamentais:
 - SDA – *Serial Data* (A4 no Arduino)
 - SCL – *Serial Clock* (A5 no Arduino)

} É muito simples adicionar dispositivos ao bus I²C



☐ Estrutura e aplicações

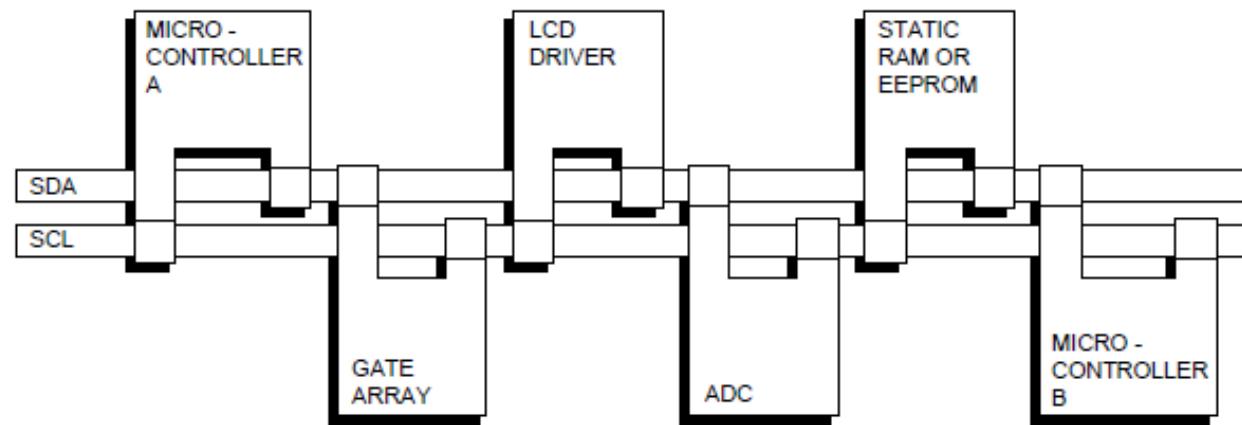
- Cada dispositivo no *bus* tem o seu próprio endereço único, seja ele um microcontrolador, uma memória, um controlador de LCD, uma interface de teclado, etc.





□ Comunicação no bus

- A comunicação no bus é feita segundo o princípio de *Master-Slave*.
- Um *Master*, é qualquer dispositivo que inicia uma transferência de dados sobre o *bus* e que gera o sinal de *clock* para tal.
- Um *Slave*, é qualquer dispositivo que esteja a ser endereçado.





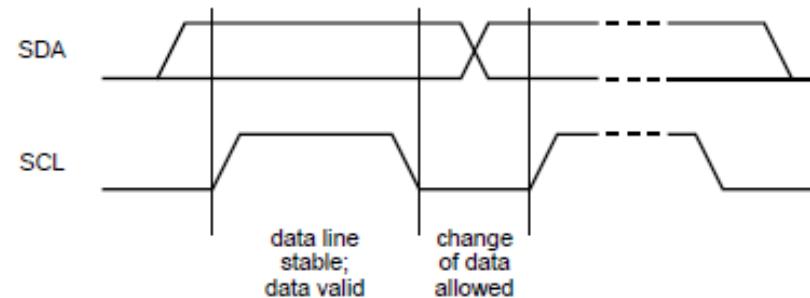
□ Comunicação no *bus*

- Terminologia:
 - Transmissor: o dispositivo que envia dados para o *bus*
 - Recetor: o dispositivo que recebe dados do *bus*
 - *Master*: o dispositivo que inicia a transferência de dados, gera o sinal de *clock* e termina a transferência
 - *Slave*: o dispositivo endereçado pelo *Master*
 - *Multi-master*: Mais do que um *Master* pode tentar controlar o *bus* em simultâneo, sem corromper a mensagem



□ Comunicação no *bus*

- Terminologia:
- Arbitragem: Procedimento para garantir que, se mais do que um *Master* tentar controlar o *bus*, apenas um é autorizado a fazê-lo e a mensagem não fica corrompida. Resolvido entre *Masters*.
- Os dados na linha SDA só podem mudar quando a linha SCL está a “0”. Enquanto SCL estiver a “1”, SDA tem que estar estável.



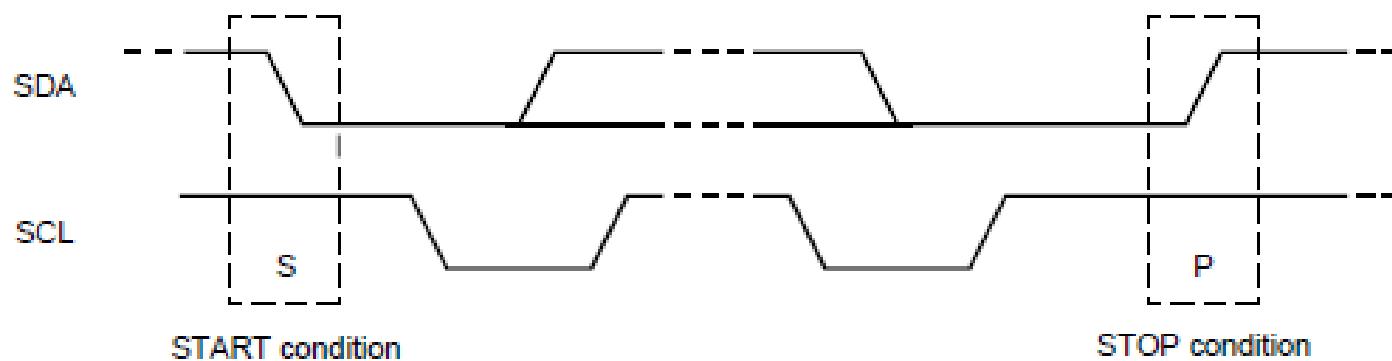


□ Comunicação no bus

- Em repouso, as linhas SDA (A4) e SCL (A5) estão ao nível lógico “1”.
- Todas as transações são iniciadas com START (S) e terminadas com STOP (P). Estas condições são sempre geradas pelo *Master*.

START (S): SCL está a “1” e ocorre uma transição descendente em SDA.

STOP (P): SCL está a “1” e ocorre uma transição ascendente em SDA.





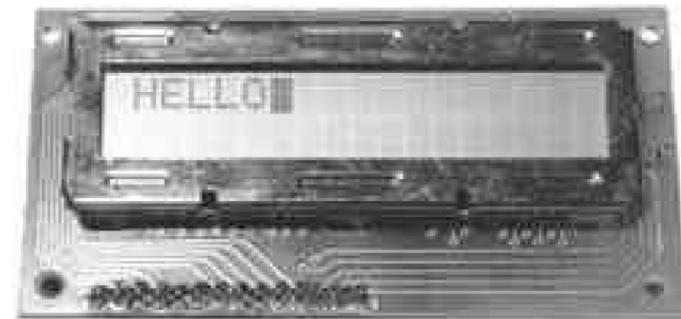
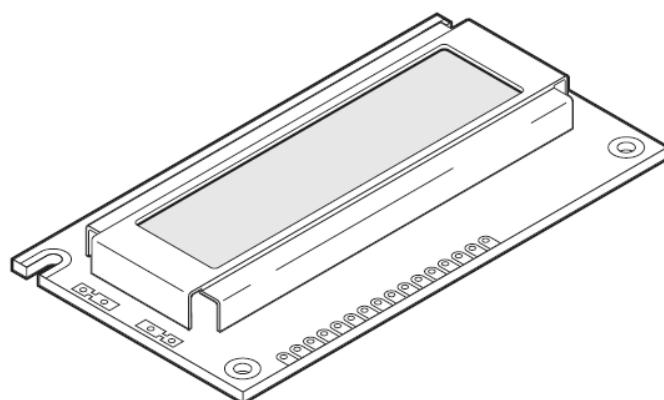
□ Comunicação no *bus*

- Entre as condições de S e P, o *bus* estará ocupado, voltando a estar livre após P.
- Por cada ciclo de SCL, é enviado um bit de informação. A transmissão é orientada ao byte (8 bits). Após o 8.^º bit, é recebido um bit de *acknowledge* (ACK) por parte do recetor, informando o transmissor de que pode enviar outro byte, porque o anterior foi corretamente recebido. Para este efeito, o transmissor liberta o SDA, de tal forma que o recetor o coloca a “0” no 9.^º *clock*. Se SDA for colocado a “1”, significa *Not acknowledge* (NACK), ou seja, que a transmissão do byte anterior não foi bem sucedida.



□ Introdução

- Utilizados em equipamentos comerciais e industriais onde os requisitos de visualização são relativamente simples, em oposição a monitores, por exemplo.
- A interface é bastante simples, nomeadamente para interligação a microcontroladores

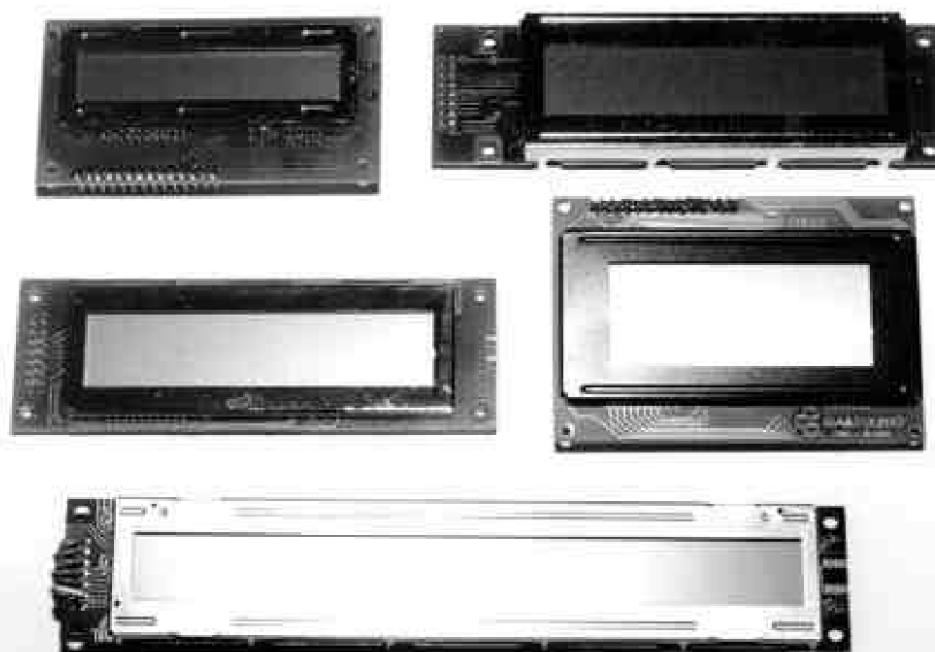




□ Formas e tamanhos

- Apesar de só se poderem visualizar caracteres, as formas e tamanhos são variadas, podendo encontrar-se comercialmente *displays* com:
 - 8, 16, 20, 24, 32 ou 40 caracteres de comprimento
 - 1, 2 ou 4 linhas de altura

Habitualmente, estes *displays* têm uma luz para retroiluminação (*backlight*) e 14 pinos de interface física: 8 linhas de dados, 3 linhas de controlo e 3 linhas de alimentação.





□ Pinos para interface física

- Dados: $D_7 \dots D_0$
- Controlo: *Register Select* (RS), *Read/Write* (R/\overline{W}) e *Enable* (E)
- Alimentação: *Ground* (V_{SS}), positivo (V_{DD}) e contraste (V_{EE})

RS – “0”: A informação enviada para o *display* é um comando e a informação lida indica o estado do *display*.

“1”: A informação trocada entre o *display* e o controlador é tomada como dados (caracteres).



□ Pinos para interface física

R/ \overline{W} – “0”: Escrita de comando ou caracteres.

“1”: Leitura de caracteres ou informação de estado.

E – Usado para iniciar a transferência de comando ou dados. Para escrever no *display*, a transferência dá-se na transição descendente de E. Para ler, a informação fica disponível após a transição descendente e ficará disponível até ao próximo flanco descendente.

Os dados podem ser transferidos a 8 ou a 4 bits. Neste último caso, só as linhas D₇ a D₄ são usadas. A intenção é a de poupar pinos de I/O ao dispositivo interlocutor com o *display*.



□ Comandos possíveis para o *display*

Em modo de “comando” ($RS = 0$), pode introduzir-se um de oito tipos de comando possíveis:

- ***Clear display*** – Limpar o *display* (tornar vazio)
- ***Display and Cursor Home*** – Colocar o cursor na posição inicial
- ***Character Entry Mode*** – Modo de escrita (para a esquerda ou direita)
- ***Display ON/OFF & Cursor*** – Aparência do *display* e do cursor
- ***Display/Cursor Shift*** – Comportamento do cursor ou do *display* com a escrita



□ Comandos possíveis para o *display* (continuação)

- ***Function Set*** – Configuração básica de funcionamento do *display*
- ***Set CGRAM Address*** – Endereçar RAM geradora de caracteres (do utilizador)
- ***Set Display Address*** – Posicionar o cursor no *display*

A seguir a cada comando deve dar-se um tempo de guarda por forma a garantir que o *display* processa correta e completamente o comando atual, antes de receber outro.



□ Comandos para o *display* e sua codificação

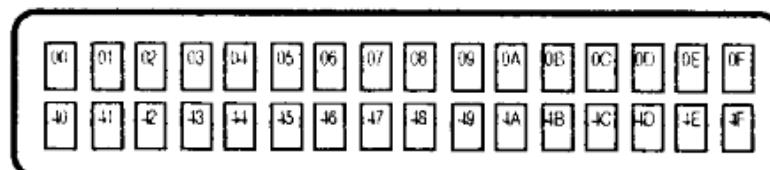
Command	Binary								Hex
	D7	D6	D5	D4	D3	D2	D1	D0	
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	1 / D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D / C	R / L	x	x	10 to 1F
Function Set	0	0	1	8 / 4	2 / 1	10 / 7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

1 / D: 1=Increment*, 0=Decrement **R / L:** 1=Right shift, 0=Left shift
S: 1=Display shift on, 0=Off* **8 / 4:** 1=8-bit interface*, 0=4-bit interface
D: 1=Display on, 0=Off* **2 / 1:** 1=2 line mode, 0=1 line mode*
U: 1=Cursor underline on, 0=Off* **10 / 7:** 1=5x10 dot format, 0=5x7 dot format*
B: 1=Cursor blink on, 0=Off*
D / C: 1=Display shift, 0=Cursor move x = Don't care * = Initialization settings

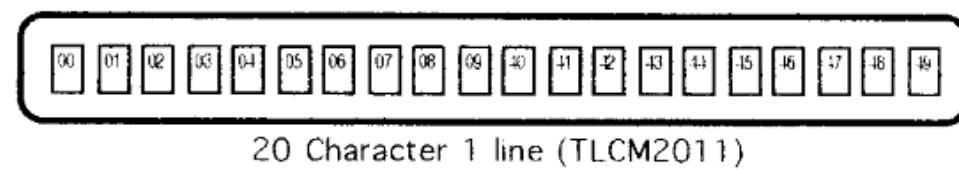


☐ Endereçamento dos caracteres no *display*

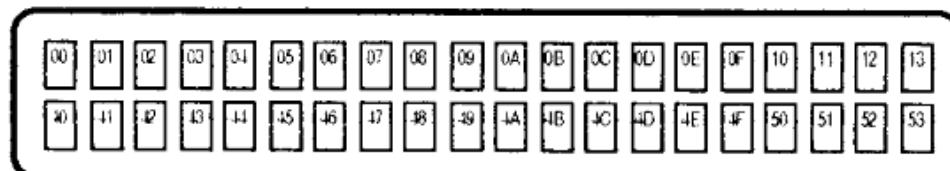
- Dependendo do *display*, os caracteres têm o endereçamento mostrado de seguida. Aquele que nos interessa é o de 2 linhas e 16 colunas.



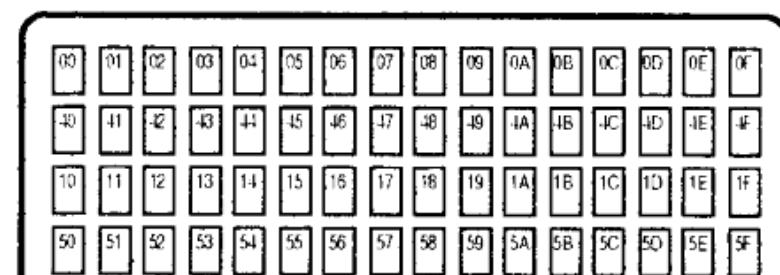
16 Character 2 line (TLCM1621 or LM016L)



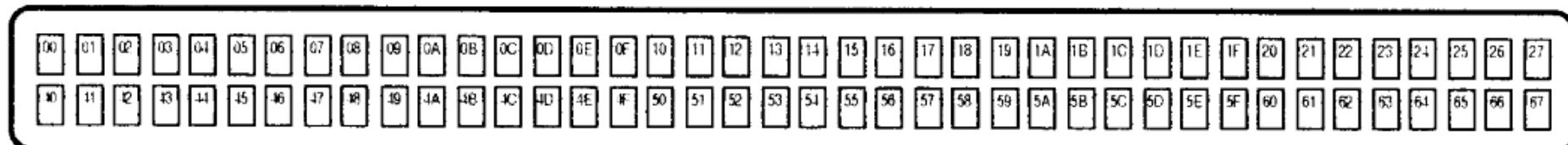
20 Character 1 line (TLCM2011)



20 Character 2 line (LM032L)



16 Character 4 line (SMC1640A or LM041L)



40 Character 2 line (TLCM4021 or LM018L)



□ Adaptação do *display* de paralelo para I²C

- Normalmente, o *display* em si possui aquilo que se denomina por “interface paralela”, a qual é composta pelos 14 pinos de interface física já mencionados. Contudo, esta interface pode ser convertida para I²C, recorrendo a um módulo baseado no circuito integrado PCF8574.



Nesta situação, a comunicação com o *display* é feita no modo de 4 bits, também já mencionado. Para inicializar o *display*, deve ter-se esta característica em conta. O procedimento de inicialização, para o *display* com conversor, está descrito nas folhas do Prof. Jorge Pais.



☐ Display LCD (*seeed*)

- O *display* que se mostra de seguida tem a interface I²C já incorporada e funciona segundo uma comunicação a 8 bits. Endereço típico: **0x3E**.



- O envio de «comando» para o *display* processa-se de forma equivalente a uma escrita I²C no registo com endereço **0x80**, enquanto que «dados» devem ser escritos no endereço **0x40**.



☐ Envio de informação para o *display* LCD (seeed)

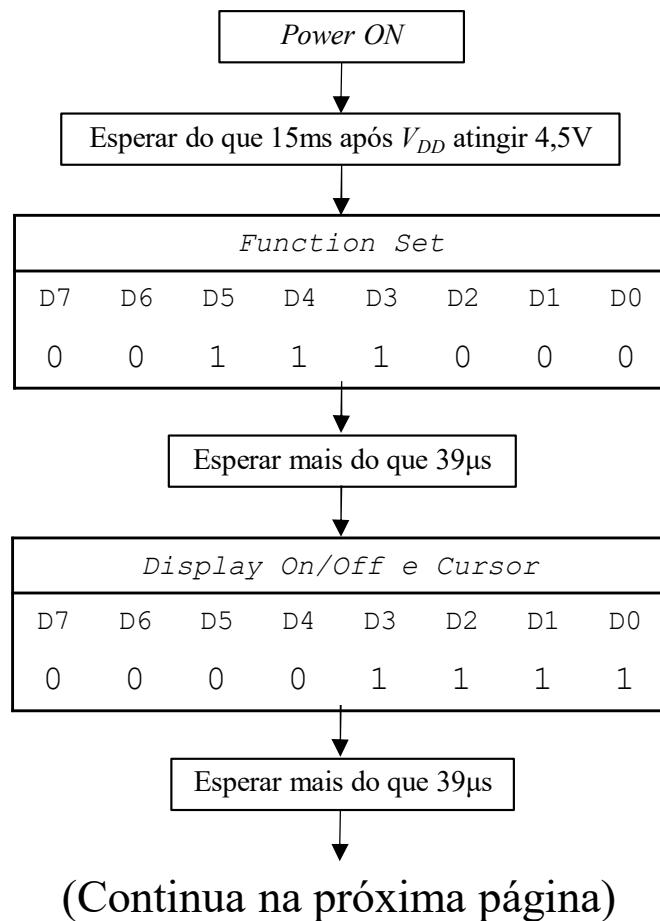
- Para enviar um comando, torna-se útil implementar uma função auxiliar de acesso básico, ao usar o tipo de *display* mostrado atrás.

```
void escreverComando8(byte oitoBits){  
    Wire.beginTransmission(0x3E);  
    Wire.write(0x80);  
    Wire.write(oitoBits);  
    Wire.endTransmission();  
}
```

No que respeita aos comandos, existe total compatibilidade entre o *display* com adaptador I²C e o presente *display*, com interface I²C nativa. Contudo, procedimento de inicialização é diferente, sendo indicado de seguida.



□ Procedimento de inicialização do *display LCD* (*seeed*)



Em todas as situações da inicialização,

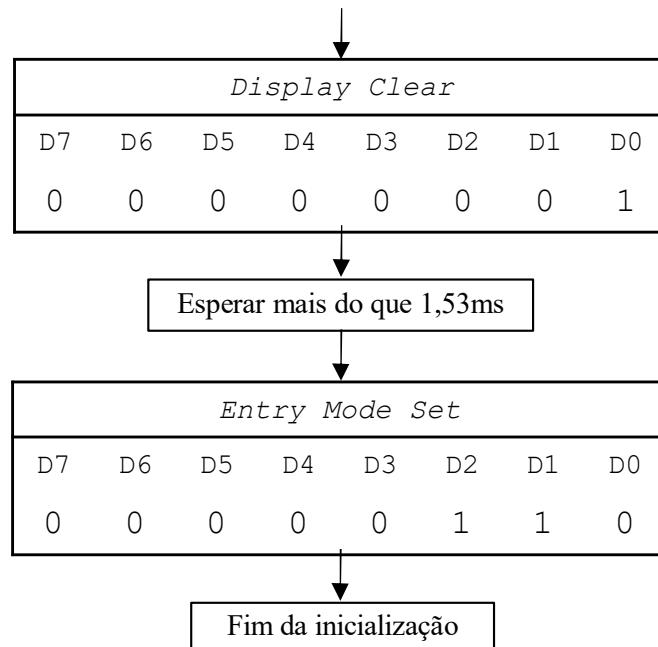
$$RS = 0 \text{ e } R/W = 0.$$

- *Function Set*: configurar para ter uma interface a 8 bits e duas linhas de caracteres de 5×7 pixels;
- *Display On/Off e Cursor*: Fazer *display On*, *cursor ON* e *blink ON*.



□ Procedimento de inicialização do *display* LCD (*seeed*)

(Continuação da página anterior)



Em todas as situações da inicialização,
 $RS = 0$ e $R/W = 0$.

- *Display Clear*: Limpar todo o conteúdo textual do *display*;
- *Entry Mode Set*: Incrementar a posição do cursor, sem fazer *shift* ao texto no *display*.



□ Codificação dos caracteres típicos disponíveis no *display LCD*

	Upper 4 bits	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	Lower 4 bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	CG RAM (1)	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
1	CG RAM (2)	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	1111
2	CG RAM (3)	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	1111	1111
3	CG RAM (4)	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	1111	1111	1111
4	CG RAM (5)	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	1111	1111	1111	1111
5	CG RAM (6)	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	1111	1111	1111	1111	1111
6	CG RAM (7)	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	1111	1111	1111	1111	1111	1111
7	CG RAM (8)	0111	1000	1001	1010	1011	1100	1101	1110	1111	1111	1111	1111	1111	1111	1111	1111
8	CG RAM (1)	1000	1001	1010	1011	1100	1101	1110	1111	1111	1111	1111	1111	1111	1111	1111	1111
9	CG RAM (2)	1001	1010	1011	1100	1101	1110	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111
A	CG RAM (3)	1010	1011	1100	1101	1110	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111
B	CG RAM (4)	1011	1100	1101	1110	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111
C	CG RAM (5)	1100	1101	1110	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111
D	CG RAM (6)	1101	1110	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111
E	CG RAM (7)	1110	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111
F	CG RAM (8)	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111

Alguns destes caracteres estão disponíveis diretamente pela introdução pelo seu código em formato `char`, por exemplo, 'a'. Outros, menos usuais, podem obter-se através da introdução do valor em hexadecimal, por exemplo, π (código `0xF7`).



☐ Funções de envio de comando e de dados a implementar no Arduino

- void escreverComando4 (byte quatroBits)
- void escreverComando8 (byte oitoBits)
- void escreverDados4 (byte quatroBits)
- void escreverDados8 (byte oitoBits)

**Funções de
acesso básico**

- void displayInit ()
- void displayClear ()
- void displaySetCursor (byte linha, byte coluna)
- void displayCursorOffBlinkOff ()
- void displayCursorOnBlinkOn ()
- void displayCursorHome ()

Comandos

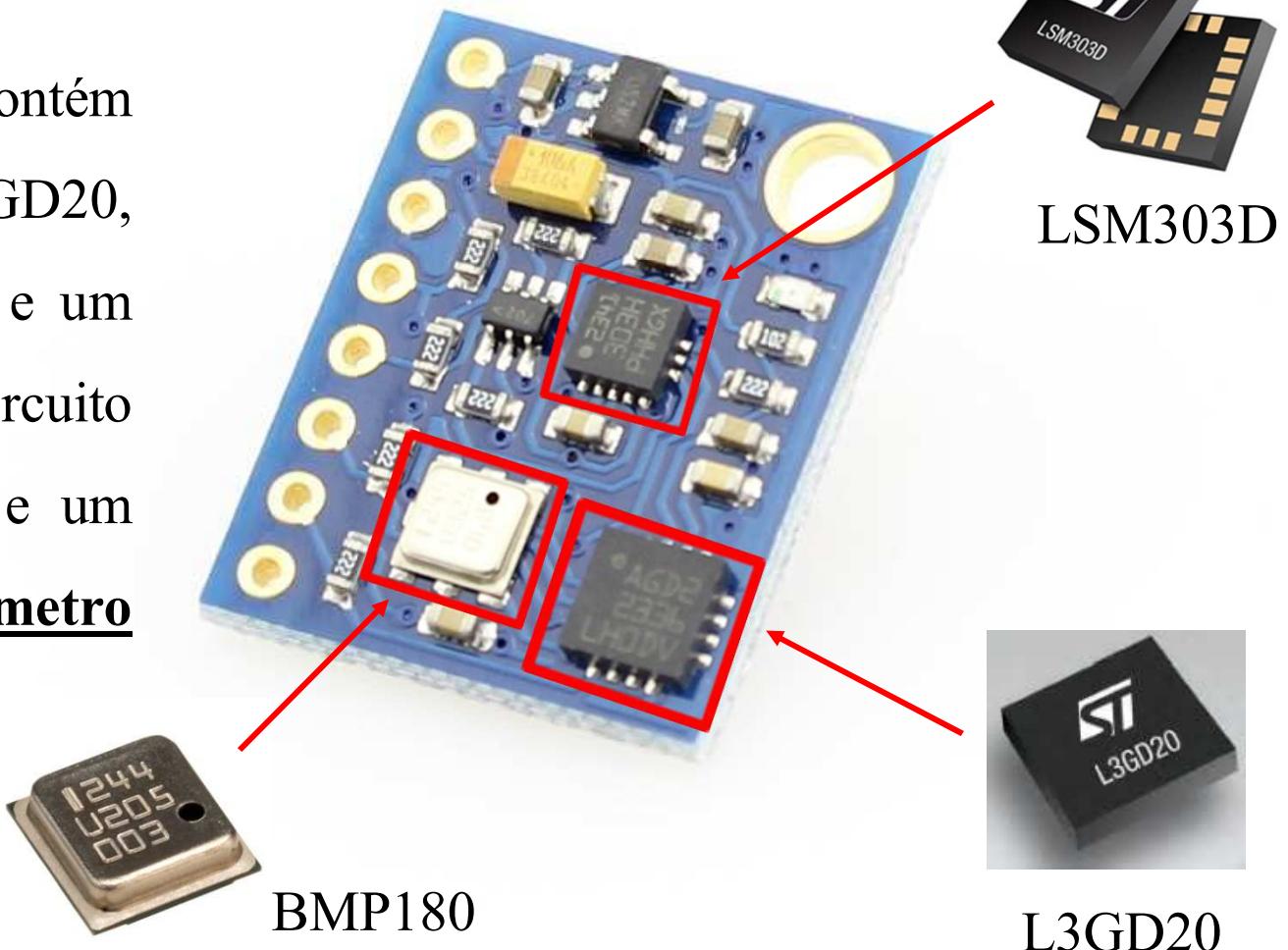
- void displayPrintChar (char c)
- void displayPrintString (String s)

Dados



□ Placa IMU 10 DOF com L3GD20, LSM303D e BMP180

A placa mostrada contém um giroscópio no L3GD20, um acelerómetro e um magnetómetro, no circuito integrado LSM303D, e um barómetro e termómetro digital no BMP180.





☐ Circuitos integrados L3GD20, LSM303D e BMP180



L3GD20



LSM303D



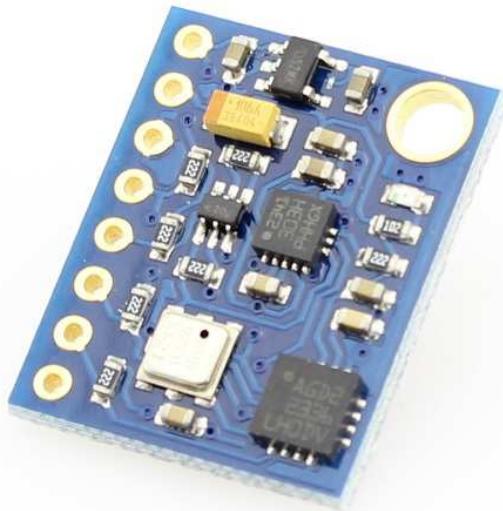
BMP180

O conjunto destes dispositivos consiste num IMU (*Inertial Measurement Unit*).

Cada um dos sensores 3D (giroscópio, acelerómetro e magnetómetro), fornece informação sobre três eixos no espaço, totalizando nove (9) informações distintas: *nine degrees of freedom* – 9 DOF. O barómetro (sensor de pressão) fornece mais uma (1) informação, relativa à pressão atmosférica, permitindo obter da placa um total de dez (10) informações diferentes: 10 DOF.



□ Aplicações e interface de alimentação e dados



Esta placa é utilizada em aplicações de *gaming*, em dispositivos de *input* de realidade virtual, interfaces homem-máquina (MMI - *man-machine interface*), sistemas de navegação por GPS, dispositivos robóticos e *drones*.

- VIN – **Alimentação** (3.3 V).
- GND – **Referência elétrica** (massa).
- SCL – **Clock** I²C.
- SDA – **Dados** I²C.



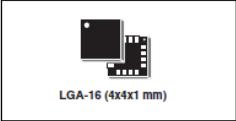
□ Datasheets do L3GD20 e LSM303D

L3GD20

STMicroelectronics

MEMS motion sensor:
three-axis digital output gyroscope

Datasheet - production data



LGA-16 (4x4x1 mm)

Applications

- Gaming and virtual reality input devices
- Motion control with MMI (man-machine interface)
- GPS navigation systems
- Appliances and robotics

Description

The L3GD20 is a low-power three-axis angular rate sensor. It includes a sensing element and an IC interface capable of providing the measured angular rate to the external world through a digital interface (I²C/GPI).

The sensing element is manufactured using a dedicated micro-machining process developed by STMicroelectronics to produce inertial sensors and actuators on silicon wafers.

The IC interface is manufactured using a CMOS process that allows a high level of integration to design a dedicated circuit which is trimmed to better match the sensing element characteristics.

The L3GD20 has a full scale of ±250/±500/±2000 dps and is capable of measuring rates with a user-selectable bandwidth.

Wide supply voltage: 2.4 V to 3.6 V

Low voltage-compatible I/Os (1.8 V)

Embedded power-down and sleep mode

Embedded temperature sensor

Embedded FIFO

High shock survivability

Extended operating temperature range (-40 °C to +85 °C)

ECOPACK®, RoHS and "Green" compliant

Table 1. Device summary

Order code	Temperature range (°C)	Package	Packing
L3GD20	-40 to +85	LGA-16 (4x4x1 mm)	Tray
L3GD20TR	-40 to +85	LGA-16 (4x4x1 mm)	Tape and reel

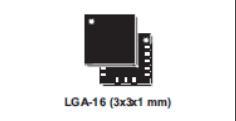
February 2012 DocID022116 Rev 2 1/44
This is information on a product in full production. www.st.com

LSM303D

STMicroelectronics

Ultra-compact high-performance eCompass module:
3D accelerometer and 3D magnetometer

Datasheet - production data



LGA-16 (3x3x1 mm)

Features

- 3 magnetic field channels and 3 acceleration channels
- ±2/±4/±8/±12 gauss magnetic full scale
- ±2/±4/±8/±16 g linear acceleration full scale
- 16-bit data output
- SPI / I²C serial interfaces
- Analog supply voltage 2.16 V to 3.6 V
- Power-down mode / low-power mode
- Programmable interrupt generators for free-fall, motion detection and magnetic field detection
- Embedded temperature sensor
- Embedded FIFO
- ECOPACK®, RoHS and "Green" compliant

Applications

- Tilt-compensated compasses
- Map rotation
- Position detection
- Motion-activated functions
- Free-fall detection
- Click/double-click recognition
- Pedometers
- Intelligent power saving for handheld devices

Table 1. Device summary

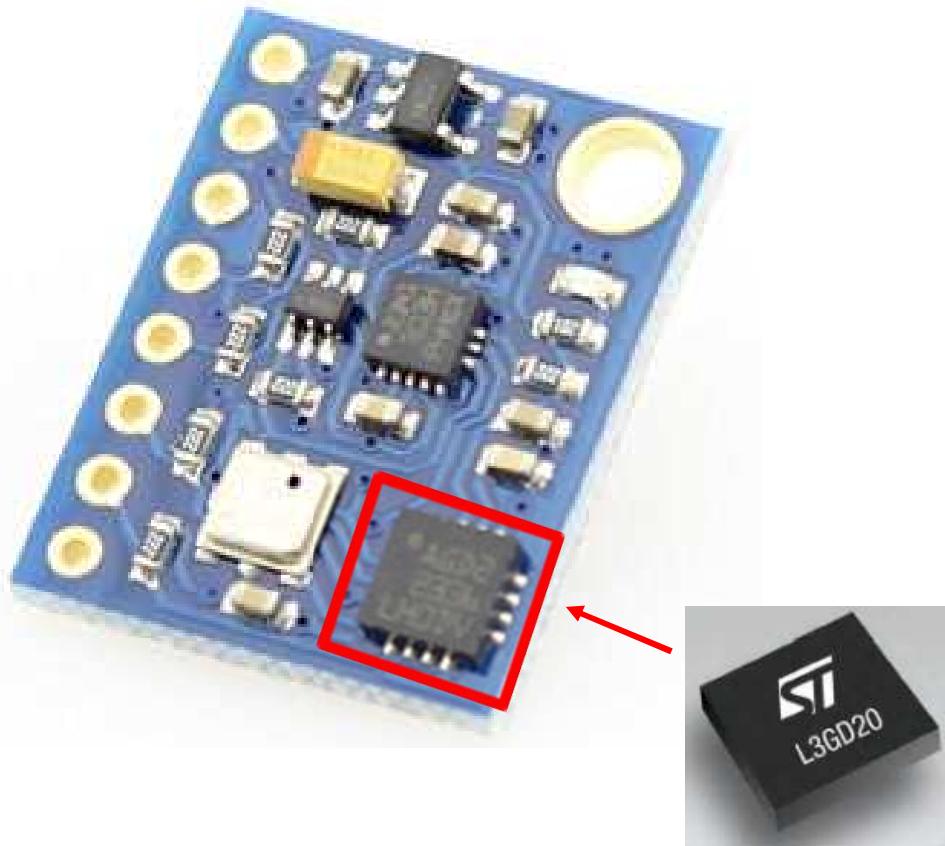
Part number	Temperature range (°C)	Package	Packaging
LSM303D	-40 to +85	LGA-16	Tray
LSM303DTR	-40 to +85	LGA-16	Tape and reel

November 2013 DocID023312 Rev 2 1/32
This is information on a product in full production. www.st.com

A utilização dos sensores 3D deve ser feita recorrendo às informações que constam nos *datasheets* do fabricante, relativamente às características e aos procedimentos a adotar para obter as informações sob os eixos X, Y e Z.



□ Introdução



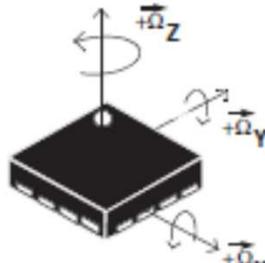
L3GD20

L3GD20 – Giroscópio digital com três eixos (X , Y e Z), com interface I²C. Permite a medição da **velocidade angular** ($^{\circ}/s$ ou rad/s), sob cada um dos eixos.

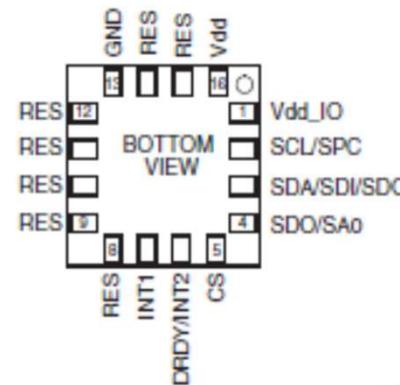
O elemento sensor é fabricado segundo a tecnologia MEMS (*Micro Electro Mechanical System*), tratando-se de um dispositivo mecânico inercial, sensível à rotação.



□ Interface física e diagrama de blocos



(TOP VIEW)
DIRECTIONS OF THE
DETECTABLE
ANGULAR RATES



Pinos do dispositivo

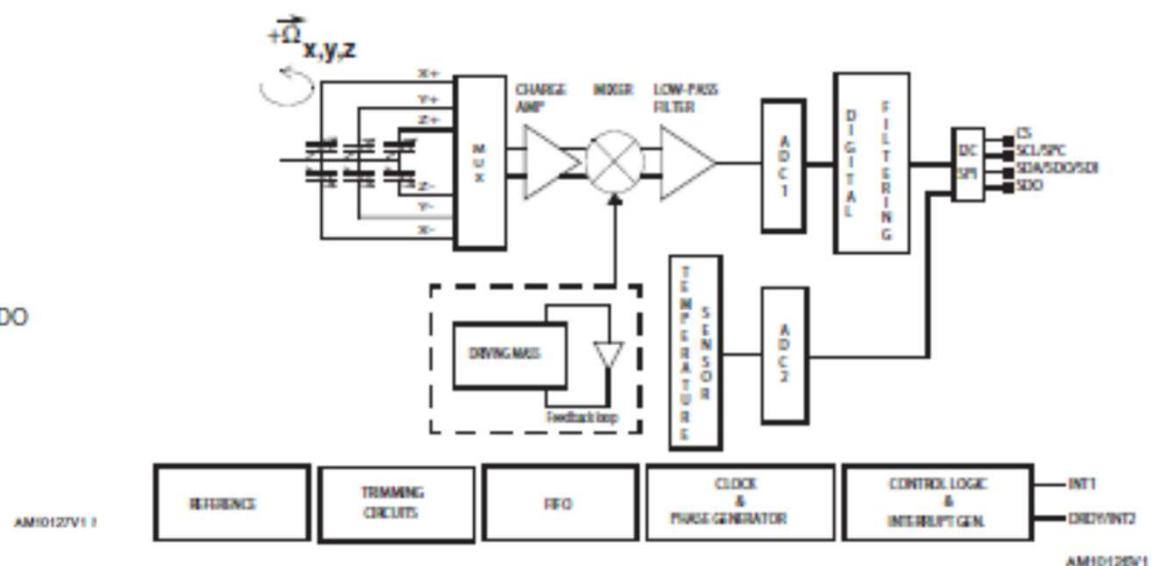
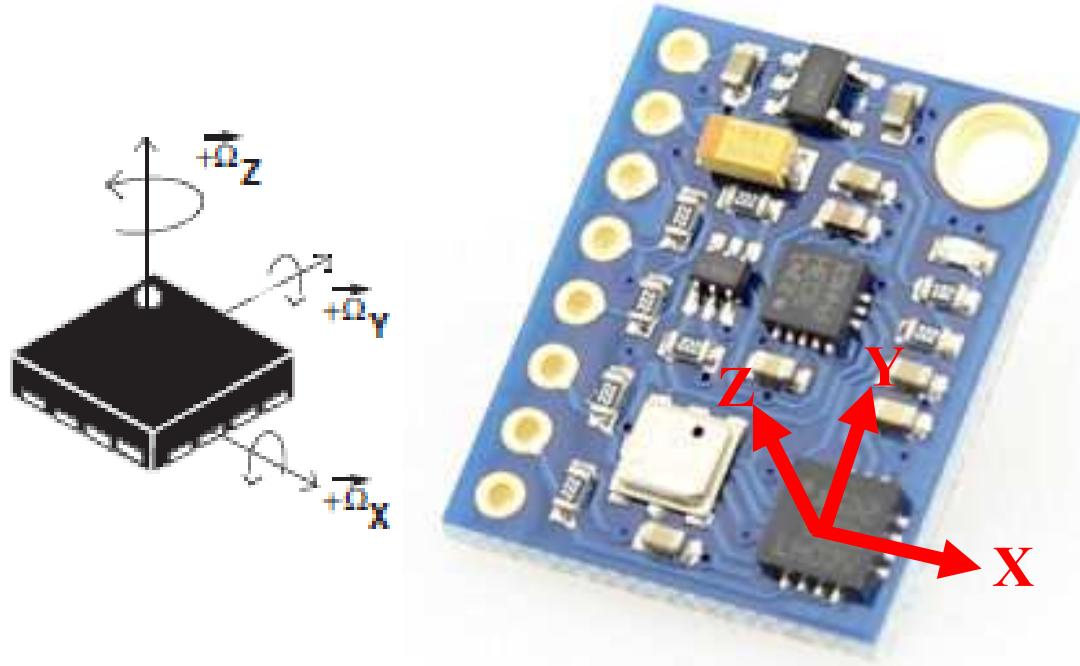


Diagrama de blocos

□ Princípio de funcionamento



Internamente, o dispositivo é mantido a vibrar a uma frequência conhecida, e, através da perturbação dessa vibração, pode estimar-se a velocidade angular sob cada um dos três eixos X , Y e Z .

Esta informação é fornecida considerando-se a rotação no sentido oposto ao dos ponteiros do relógio, estando de frente para o respetivo eixo considerado, obtendo-se Ω_X , Ω_Y e Ω_Z . A sensibilidade é configurável, de acordo com três fins de escala.



□ Características e utilização

- A interligação ao Arduino torna-se extremamente simples, graças à interface I²C, em que o endereço típico do sensor é 0x6B (ou 0x69).
- O L3GD20 dispõe de várias frequências de corte para filtragem digital dos dados adquiridos, podendo usar-se os *defaults*.
- Pode usar-se uma infraestrutura em FIFO, com 32 *slots* de 16 bits, para cada um dos três canais, para ajudar à eficiência energética do dispositivo, acordando-o de acordo com um dado critério, para extração dos dados.
- Existem três fins de escala de sensibilidade: 250 °/s, 500 °/s e 2000 °/s.



□ Terminologia

- Sensibilidade

Um giroscópio é um dispositivo que produz uma leitura de velocidade angular, para uma rotação em torno de um eixo, de acordo com o sentido oposto ao dos ponteiros do relógio. A sensibilidade é o ganho do sensor e é determinado aplicando-se uma determinada velocidade angular. Este valor altera-se muito pouco em função da temperatura e do tempo de uso.

O **fator de sensibilidade** varia de acordo com o fim de escala escolhido, sendo de $8,75 \times 10^{-3}$ ($^{\circ}/s$)/LSB, $17,50 \times 10^{-3}$ ($^{\circ}/s$)/LSB e 70×10^{-3} ($^{\circ}/s$)/LSB, para as escalas de **250** $^{\circ}/s$, **500** $^{\circ}/s$ e **2000** $^{\circ}/s$, respetivamente.



□ Terminologia

- *Zero-rate level*

O *Zero-rate level* descreve o sinal de saída no caso de não haver movimento angular. O *Zero-rate level* de sensores de precisão MEMS é, em certa medida, o resultado dos esforços a que o sensor possa ter sido sujeito, e o seu nível pode variar após a soldadura do sensor numa placa de circuito impresso, ou após a exposição a um esforço mecânico intenso. Este valor altera-se muito pouco em função da temperatura e do tempo de uso.

O *Zero-rate level* também depende das escala e é de $\pm 10^{\circ}/s$, $\pm 15^{\circ}/s$ e de $\pm 75^{\circ}/s$, respetivamente para cada uma das escalas já indicadas.



□ Modos de operação

O L3GD20 fornece três modos de operação (*operating modes*):

- *Power-down mode* (PD = 0)
- *Sleep mode* (PD = 1)
- *Normal mode* (PD = 1)

Após a aplicação de alimentação, segue-se um procedimento de arranque durante 10 ms. Após o arranque, o dispositivo fica automaticamente configurado em *Power-down mode*, pelo que tem que se configurar para sair deste modo, enviando um comando a 8 bits no CTRL_REG1. Os modos de operação implicam o consumo, sendo 5 µA, 2 mA e 6,1 mA, respetivamente.



☐ Registros do L3GD20

Table 17. Register address map

Name	Type	Register address		Default
		Hex	Binary	
Reserved	-	00-0E	-	-
WHO_AM_I	r	0F	000 1111	11010100
Reserved	-	10-1F	-	-
CTRL_REG1	rw	20	010 0000	00000111
CTRL_REG2	rw	21	010 0001	00000000
CTRL_REG3	rw	22	010 0010	00000000
CTRL_REG4	rw	23	010 0011	00000000
CTRL_REG5	rw	24	010 0100	00000000
REFERENCE	rw	25	010 0101	00000000
OUT_TEMP	r	26	010 0110	output
STATUS_REG	r	27	010 0111	output
OUT_X_L	r	28	010 1000	output
OUT_X_H	r	29	010 1001	output
OUT_Y_L	r	2A	010 1010	output
OUT_Y_H	r	2B	010 1011	output
OUT_Z_L	r	2C	010 1100	output
OUT_Z_H	r	2D	010 1101	output
FIFO_CTRL_REG	rw	2E	010 1110	00000000
FIFO_SRC_REG	r	2F	010 1111	output
INT1_CFG	rw	30	011 0000	00000000
INT1_SRC	r	31	011 0001	output
INT1_TSH_XH	rw	32	011 0010	00000000
INT1_TSH_XL	rw	33	011 0011	00000000
INT1_TSH_YH	rw	34	011 0100	00000000
INT1_TSH_YL	rw	35	011 0101	00000000
INT1_TSH_ZH	rw	36	011 0110	00000000
INT1_TSH_ZL	rw	37	011 0111	00000000
INT1_DURATION	rw	38	011 1000	00000000

Os registos identificados como «Reserved» não devem ser alterados. A escrita nestes registos pode provocar dano permanente no dispositivo.

O conteúdo dos registos é preenchido automaticamente, e reposto aquando do *boot*, (quando recebe alimentação – *powered up*) contendo os valores de calibração de fábrica.



☐ Registros de controlo do L3GD20

- WHO_AM_I – Identificação do sensor (*default*: 0xD4 ou 0xD3).

Table 18. WHO_AM_I register

1	1	0	1	0	1	0	0
---	---	---	---	---	---	---	---

- CTRL_REG1 – Definição do ODR, largura de banda da filtragem, *power-down* e *enable* ao eixos de medição de velocidade angular (ver tabelas 20, 21 e 22).

Table 19. CTRL_REG1 register

DR1	DR0	BW1	BW0	PD	Zen	Xen	Yen
-----	-----	-----	-----	----	-----	-----	-----

- CTRL_REG2 – Definição do modo e da frequência da filtragem passa-alto (pode usar-se *default* – ver tabelas 24, 25 e 26).

Table 23. CTRL_REG2 register

0 ⁽¹⁾	0 ⁽¹⁾	HPM1	HPM0	HPCF3	HPCF2	HPCF1	HPCF0
------------------	------------------	------	------	-------	-------	-------	-------

1. These bits must be set to '0' to ensure proper operation of the device



☐ Registros de controlo do L3GD20

- CTRL_REG3 – Definições sobre *interrupts*, nomeadamente do FIFO e características elétricas dos terminais (pode usar-se *default* – ver tabela 28).

Table 27. CTRL_REG1 register

I1_Int1	I1_Boot	H_Lactive	PP_OD	I2_DRDY	I2_WTM	I2_ORun	I2_Empty
---------	---------	-----------	-------	---------	--------	---------	----------

- CTRL_REG4 – Definição do fim de escala de medição de velocidade angular (ver tabela 30), podendo manter os restantes bits no seu *default*.

Table 29. CTRL_REG4 register

BDU	BLE	FS1	FS0	-	0 ⁽¹⁾	0 ⁽¹⁾	SIM
-----	-----	-----	-----	---	------------------	------------------	-----

1. This value must not be changed.

- CTRL_REG5 – Definições sobre *interrupts* e FIFO (pode usar-se *default* – ver tabela 32).

Table 31. CTRL_REG5 register

BOOT	FIFO_EN	--	HPen	INT1_Sel1	INT1_Sel0	Out_Sel1	Out_Sel0
------	---------	----	------	-----------	-----------	----------	----------



☐ Registros de controlo do L3GD20

- STATUS_REG – Indicadores (*flags*) sobre se existe nova informação disponível (*data available* – DA) a partir do giroscópio, ou se houve dados que foram medidos, mas que não foram entretanto lidos, tendo sido esmagados (*overrun* – OR). Estes indicadores são individualizados por eixo, havendo ainda para DA ou OR, um bit apenas que indica a ocorrência de cada uma das situações anteriores em algum dos eixos (ver tabela 38).

Table 37. STATUS_REG register

ZYXOR	ZOR	YOR	XOR	ZYXDA	ZDA	YDA	XDA
-------	-----	-----	-----	-------	-----	-----	-----



☐ Registros de dados de velocidade angular do L3GD20

- OUT_X_H e OUT_X_L – Informação a oito bits sobre a parte alta e a parte baixa do valor digitalizado da velocidade angular sob o eixo do X.
- OUT_Y_H e OUT_Y_L – Informação a oito bits sobre a parte alta e a parte baixa do valor digitalizado da velocidade angular sob o eixo do Y.
- OUT_Z_H e OUT_Z_L – Informação a oito bits sobre a parte alta e a parte baixa do valor digitalizado da velocidade angular sob o eixo do Z.

A informação obtida deverá ser **convenientemente concatenada** e **multiplicada pelo fator de sensibilidade**, de acordo com o **fim de escala** escolhido.



☐ Registo de dados de temperatura do L3GD20

Existe também um registo, a partir do qual se pode obter, indiretamente, uma estimativa da temperatura ambiente.

- OUT_TEMP – Informação a oito bits, expressa em complemento para dois, sobre a temperatura do ambiente onde o sensor está colocado.

A sensibilidade do sensor de temperatura, *versus* a informação colocada neste registo, é de -1 °C/bit (ver tabela 5 do manual do L3GD20).

Pode ser realizado um processamento empírico aproximado, para fazer corresponder o valor lido do registo, com o da temperatura ambiente.



☐ Sequência de funcionamento

1. Inicialização do dispositivo, nomeadamente, verificando a identificação do sensor e escrevendo os valores adequados nos registos que tenham que funcionar com valores diferentes dos de *default*.
2. Leitura da informação a 16 bits (inteiros com sinal), relativa à velocidade angular sob um determinado eixo (X , Y ou Z), multiplicada pelo fator de sensibilidade inerente ao fim de escala em vigor.

Se a velocidade angular for constante, a estimativa do deslocamento angular pode ser obtida fazendo o integral de Ω , num dado intervalo de tempo, segundo um eixo, *e.g.*: $\varphi_Z = \int_{t_0}^{t_1} \Omega_Z \, dt + \varphi_{Z0} = \Omega_Z \cdot (t_1 - t_0) + \varphi_{Z0}$



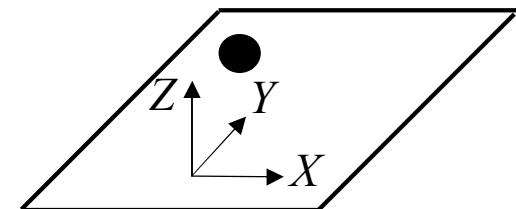
□ Funções de interação com o L3GD20

Funções de acesso básico:

- byte L3GD20_read_8bit_value(byte regAddress)
- void L3GD20_write_8bit_value(byte regAddress, byte value)

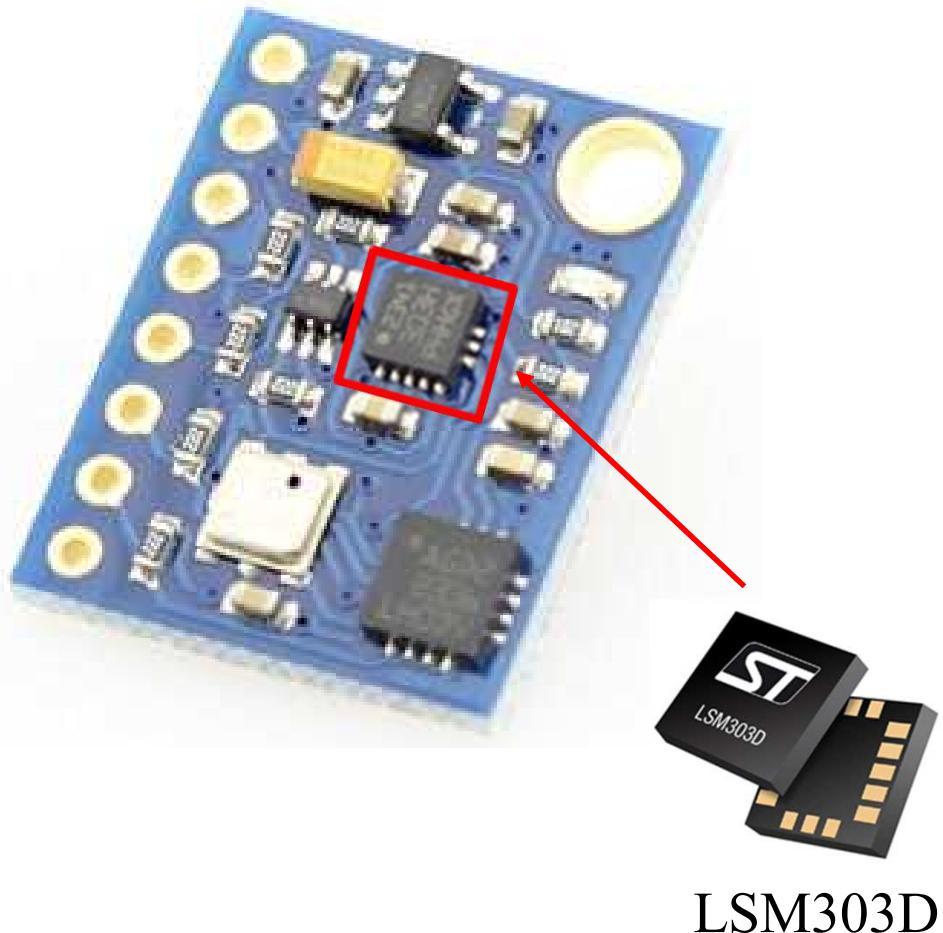
Funções de inicialização e leitura de informação:

- bool L3GD20_Init(byte senseRange)
- float L3GD20_X_channel_Read()
- float L3GD20_Y_channel_Read()
- float L3GD20_Z_channel_Read()
- byte L3GD20_Temperature_Read()





□ Introdução



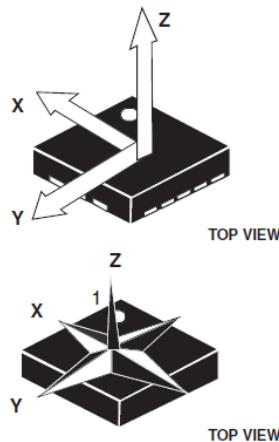
LSM303D – Acelerómetro e magnetómetro digitais com três eixos (X , Y e Z), com interface I²C. Permite a medição da aceleração (g), e da indução magnética (gauss) sob cada um dos eixos. Utilizado em bússolas compensadas pela inclinação, rotação de mapas e pedómetros.

$$1 \text{ } g = 9,8 \text{ m/s}^2 \rightarrow 1 \text{ m/s}^2 = 0,102 \text{ } g$$

$$1 \text{ gauss} = 100 \mu\text{T} \rightarrow 1 \mu\text{T} = 0,01 \text{ gauss}$$

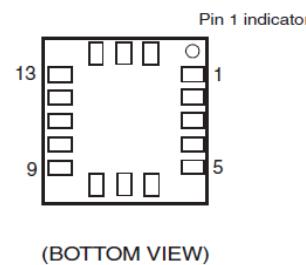


☐ Interface física e diagrama de blocos

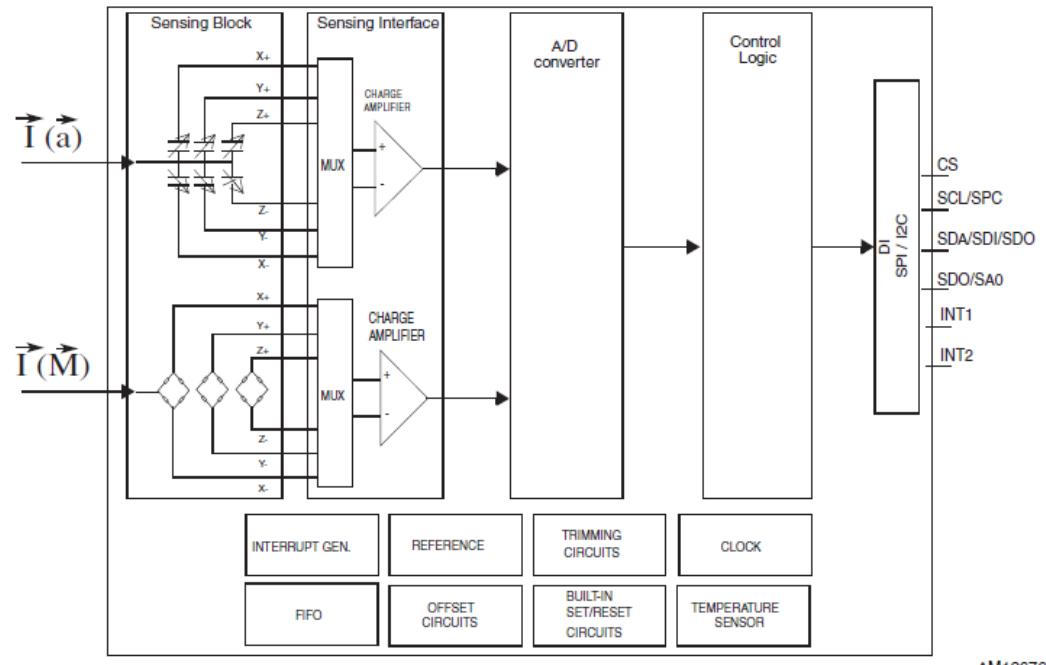


DIRECTION OF
DETECTABLE
ACCELERATIONS

DIRECTION OF
DETECTABLE
MAGNETIC FIELDS



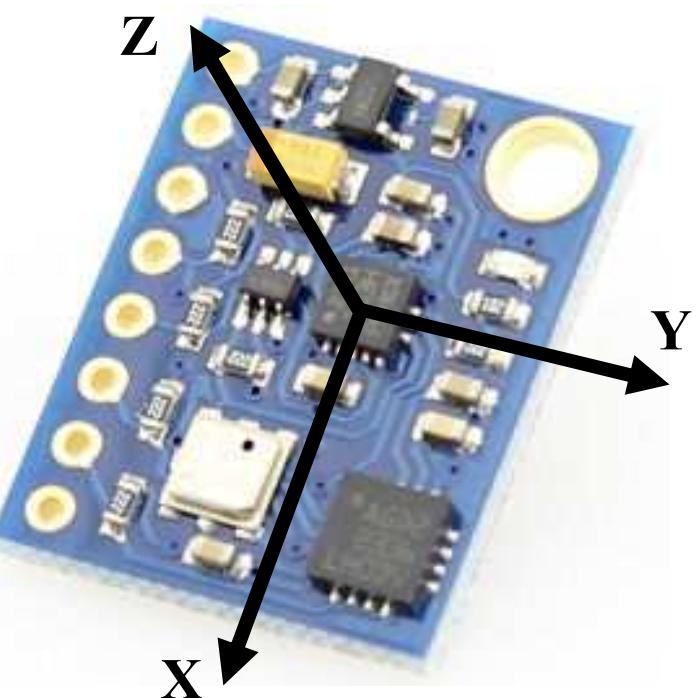
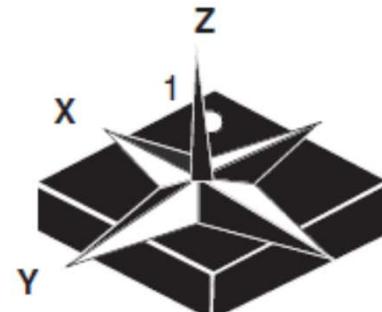
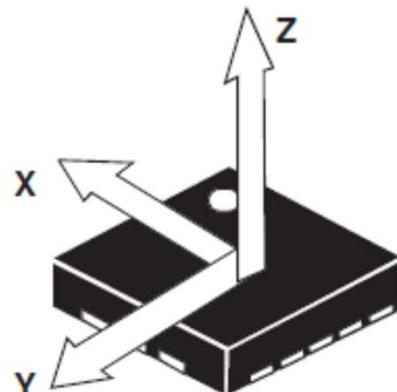
AM12677V1



Pinos do dispositivo

Diagrama de blocos

□ Eixos de funcionamento



A informação sobre a aceleração ou campo magnético é fornecida para cada eixo, considerando-se a direção e o sentido mostrados. A sensibilidade da aceleração é configurável, de acordo com cinco fins de escala e a magnética, de acordo com quatro.



□ Características e utilização

- A interligação ao Arduino torna-se extremamente simples, graças à interface I²C, em que o endereço típico do *chip* com o sensor de aceleração e o sensor magnético é 0x1D (ou 0x1E).
- O LSM303D dispõe de várias frequências de corte para filtragem digital dos dados adquiridos, podendo usar-se os *defaults*.
- Existem cinco (5) fins de escala de sensibilidade do acelerómetro: $\pm 2\text{ g}$, $\pm 4\text{ g}$, $\pm 6\text{ g}$, $\pm 8\text{ g}$, $\pm 16\text{ g}$.
- Existem quatro (4) fins de escala de sensibilidade do sensor de indução magnética: $\pm 2\text{ gauss}$, $\pm 4\text{ gauss}$, $\pm 8\text{ gauss}$ e $\pm 12\text{ gauss}$.



☐ Registros do LSM303D e respetivos endereços

Table 16. Register address map

Name	Type	Register address		Default	Comment
		Hex	Binary		
Reserved	—	00-04	—	—	Reserved
TEMP_OUT_L	r	05	000 0101	Output	
TEMP_OUT_H	r	06	000 0110	Output	
STATUS_M	r	07	000 0111	Output	
OUT_X_L_M	r	08	000 1000	Output	
OUT_X_H_M	r	09	000 1001	Output	
OUT_Y_L_M	r	0A	000 1010	Output	
OUT_Y_H_M	r	0B	000 1011	Output	
OUT_Z_L_M	r	0C	000 1100	Output	
OUT_Z_H_M	r	0D	000 1101	Output	
Reserved	—	0E	000 1110	—	Reserved
WHO_AM_I	r	0F	000 1111	01001001	
Reserved	—	10-11	—	—	Reserved
INT_CTRL_M	rw	12	001 0010	11101000	
INT_SRC_M	r	13	001 0011	Output	
INT_THS_L_M	rw	14	001 0100	00000000	
INT_THS_H_M	rw	15	001 0101	00000000	
OFFSET_X_L_M	rw	16	001 0110	00000000	
OFFSET_X_H_M	rw	17	001 0111	00000000	
OFFSET_Y_L_M	rw	18	001 01000	00000000	
OFFSET_Y_H_M	rw	19	001 01001	00000000	
OFFSET_Z_L_M	rw	1A	001 01010	00000000	
OFFSET_Z_H_M	rw	1B	001 01011	00000000	
REFERENCE_X	rw	1C	001 01100	00000000	
REFERENCE_Y	rw	1D	001 01101	00000000	
REFERENCE_Z	rw	1E	001 01110	00000000	
CTRL0	rw	1F	001 1111	00000000	
CTRL1	rw	20	010 0000	00000111	
CTRL2	rw	21	010 0001	00000000	

Table 16. Register address map (continued)

Name	Type	Register address		Default	Comment
		Hex	Binary		
CTRL3	rw	22	010 0010	00000000	
CTRL4	rw	23	010 0011	00000000	
CTRL5	rw	24	010 0100	00011000	
CTRL6	rw	25	010 0101	00100000	
CTRL7	rw	26	010 0110	00000001	
STATUS_A	r	27	010 0111	Output	
OUT_X_L_A	r	28	010 1000	Output	
OUT_X_H_A	r	29	010 1001	Output	
OUT_Y_L_A	r	2A	010 1010	Output	
OUT_Y_H_A	r	2B	010 1011	Output	
OUT_Z_L_A	r	2C	010 1100	Output	
OUT_Z_H_A	r	2D	010 1101	Output	
FIFO_CTRL	rw	2E	010 1110	00000000	
FIFO_SRC	r	2F	010 1111	Output	
IG_CFG1	rw	30	011 0000	00000000	
IG_SRC1	r	31	011 0001	Output	
IG_THS1	rw	32	011 0010	00000000	
IG_DUR1	rw	33	011 0011	00000000	
IG_CFG2	rw	34	011 0100	00000000	
IG_SRC2	r	35	011 0101	Output	
IG_THS2	rw	36	011 0110	00000000	
IG_DUR2	rw	37	011 0111	00000000	
CLICK_CFG	rw	38	011 1000	00000000	
CLICK_SRC	r	39	011 1001	Output	
CLICK_THS	rw	3A	011 1010	00000000	
TIME_LIMIT	rw	3B	011 1011	00000000	
TIME_LATENCY	rw	3C	011 1100	00000000	
TIME_WINDOW	rw	3D	011 1101	00000000	
Act_THS	rw	3E	011 1110	00000000	
Act_DUR	rw	3F	011 1111	00000000	



□ Registros do LSM303D

- Todos os registos são a 8 bits.
- Os registos identificados como «*Reserved*» não devem ser alterados. A escrita nestes registos pode provocar dano permanente no dispositivo.
- O conteúdo dos registos é preenchido automaticamente, e reposto aquando do *boot*, (quando recebe alimentação – *powered up*) contendo os valores de calibração de fábrica.



□ Terminologia – acelerómetro

- Sensibilidade do sensor de aceleração

Descreve o ganho do sensor, *i.e.* a relação entre o valor indicado pela medição e o verdadeiro valor da aceleração. Pode ser verificado, lendo os valores medidos, ao aplicar uma aceleração de $\pm 1\ g$ ($\pm 9.8\ m/s^2$), ao apontar um eixo para o centro da Terra e depois apontá-lo para o céu.

O **fator de sensibilidade da aceleração** varia de acordo com o fim de escala escolhido, sendo de **0,061 mg/LSB**, **0,122 mg/LSB**, **0,183 mg/LSB**, **0,244 mg/LSB** e **0,732 mg/LSB**, para as escalas de **$\pm 2\ g$** , **$\pm 4\ g$** , **$\pm 6\ g$** , **$\pm 8\ g$** e **$\pm 16\ g$** , respetivamente.



□ Terminologia – acelerómetro

- Nível de Zero- g

Idealmente, com o acelerómetro colocado na horizontal, a leitura, para os eixos X e Y , deve ser 0 e para Z deve ser $\pm 1\ g$. O valor 0 está ao meio da gama dinâmica do sensor, sabendo que os registos onde ficam os valores binários da medição expressam-se em complemento para 2. Se há desvio, então há erro de offset e deve-se a *stress* anterior, sofrido pelo dispositivo.

- Impulso de *set/reset* (magnetómetro)

Este impulso é uma operação automática, realizada antes de qualquer ciclo de aquisição magnética para desmagnetizar o sensor e assegurar o alinhamento dos elementos magnéticos e, portanto, a linearidade do próprio sensor.



☐ Registos de controlo mais relevantes do LSM303D – acelerómetro

- WHO_AM_I – Identificação do *chip* (acc + mag) LSM303D (*default*: 0x49).

Table 19. WHO_AM_I register

0	1	0	0	1	0	0	1
---	---	---	---	---	---	---	---

- CTRL1 – Definição do ODR, modo de *power down* e *enable* ao eixos de medição de aceleração (ver tabelas 35 e 36).

Table 34. CTRL1 register

AODR3	AODR2	AODR1	AODR0	BDU	AZEN	AYEN	AXEN
-------	-------	-------	-------	-----	------	------	------

- CTRL2 – Definição do fim de escala do acelerómetro (*pode usar-se default* para ABW, AST e SIM – ver tabelas 38, 39 e 40).

Table 37. CTRL2 register

ABW1	ABW0	AFS2	AFS1	AFS0	0 ⁽¹⁾	AST	SIM
------	------	------	------	------	------------------	-----	-----

1. This bit must be set to '0' for the correct working of the device.



☐ Registos de controlo mais relevantes do LSM303D – acelerómetro

- STATUS_A – Indicadores (*flags*) sobre se existe nova informação disponível (*data available* – DA) a partir do acelerómetro, ou se houve dados que foram medidos, mas que não foram entretanto lidos, tendo sido esmagados (*overrun* – OR). Estes indicadores são individualizados por eixo, havendo ainda para DA ou OR, um bit apenas que indica a ocorrência de cada uma das situações anteriores em algum dos eixos (ver tabela 56).

Table 55. STATUS_A register

ZYXAOR	ZAOR	YAOR	XAOR	ZYXADA	ZADA	YADA	XADA
--------	------	------	------	--------	------	------	------



☐ Registros de controlo mais relevantes do LSM303D – acelerómetro

- OUT_X_H_A e OUT_X_L_A – Informação a oito bits sobre a parte alta e a parte baixa do valor digitalizado da aceleração sob o eixo do X.
- OUT_Y_H_A e OUT_Y_L_A – Informação a oito bits sobre a parte alta e a parte baixa do valor digitalizado da aceleração sob o eixo do Y.
- OUT_Z_H_A e OUT_Z_L_A – Informação a oito bits sobre a parte alta e a parte baixa do valor digitalizado da aceleração sob o eixo do Z.

A informação obtida deverá ser **convenientemente concatenada** e **multiplicada pelo fator de sensibilidade**, de acordo com o **fim de escala** escolhido.



Sequência de funcionamento para o acelerómetro

1. Inicialização do acelerómetro, nomeadamente, verificando a identificação do circuito integrado e escrevendo os valores adequados nos registos que tenham que funcionar com valores diferentes dos de *default*, por exemplo, definindo o fim de escala.

2. Leitura da informação a 16 bits (inteiros com sinal), relativa à aceleração sob um determinado eixo (X , Y ou Z), multiplicada pelo fator de sensibilidade inerente ao fim de escala em vigor.



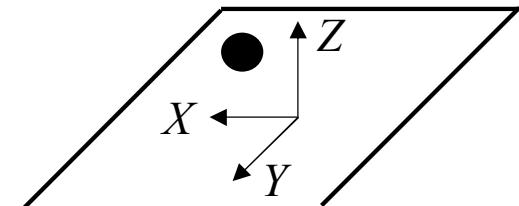
□ Funções de interação com o acelerómetro do LSM303D

Funções de acesso básico (também utilizáveis com o magnetómetro):

- byte LSM303D_read_8bit_value(byte regAddress)
- void LSM303D_write_8bit_value(byte regAddress, byte value)

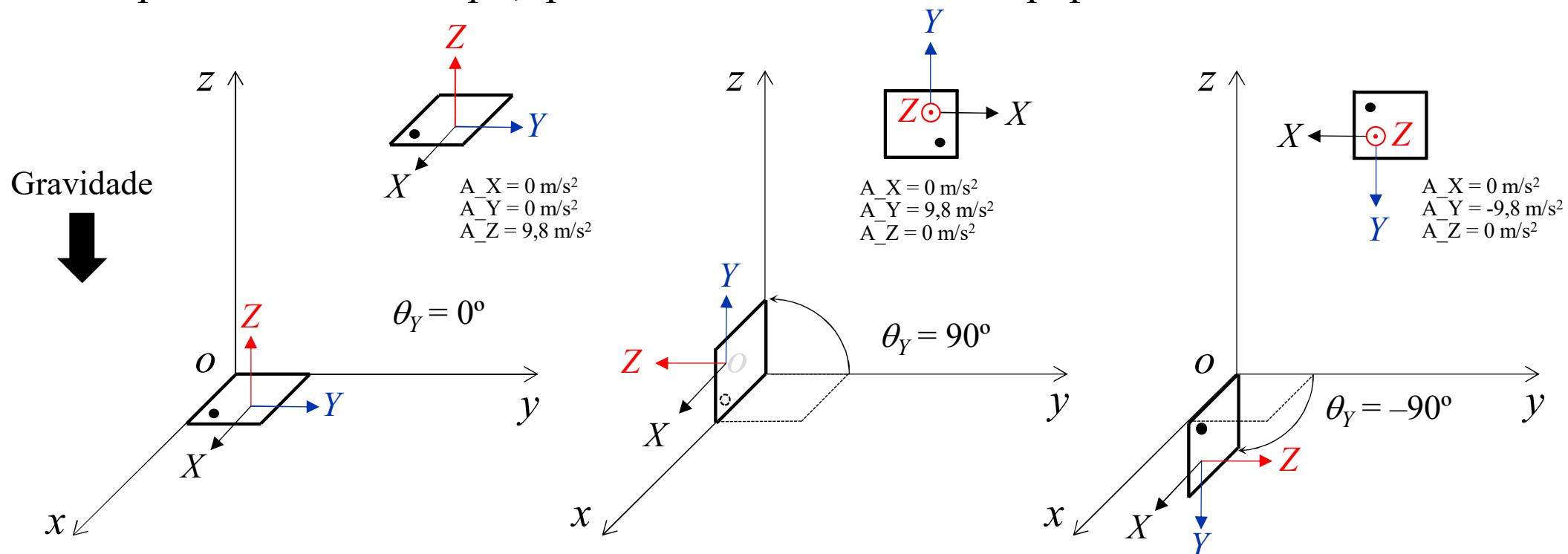
Funções de inicialização e leitura de informação do acelerómetro:

- bool LSM303D_A_Init(byte senseRange)
- float LSM303D_A_X_channel_Read()
- float LSM303D_A_Y_channel_Read()
- float LSM303D_A_Z_channel_Read()



□ Utilização do acelerómetro para medição de inclinação

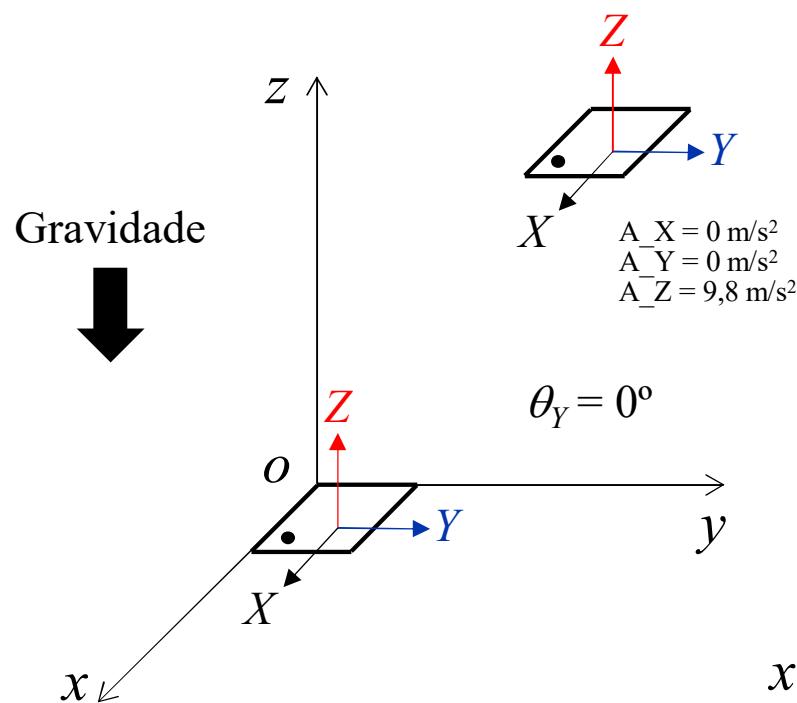
É possível usar o acelerómetro para determinar o ângulo de inclinação (*tilt angle*). É por este processo que os *smartphones* realizam a colocação do ecrã em *portrait* ou *landscape*, quando o utilizador roda o equipamento.



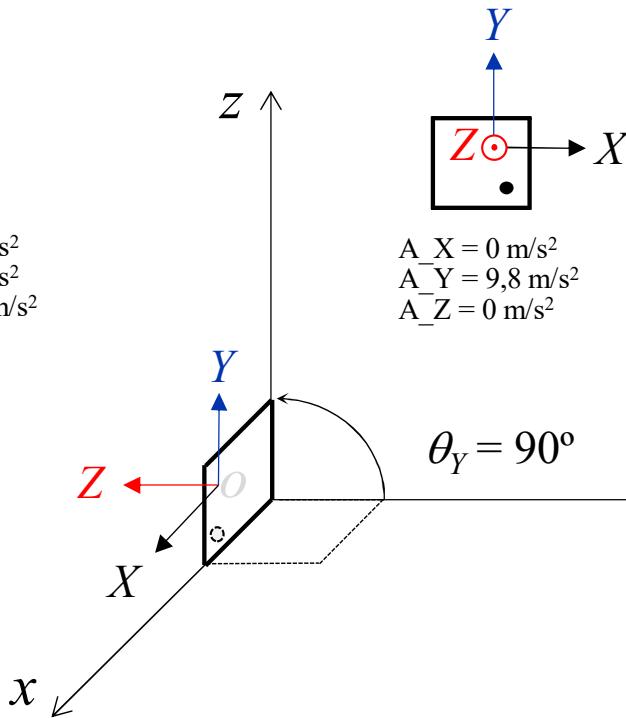
□ Utilização do acelerómetro para medição de inclinação

Usando um pouco de trigonometria, determina-se o ângulo que o acelerómetro está a fazer com o plano horizontal xoy , relativamente ao eixo y (em torno de x).

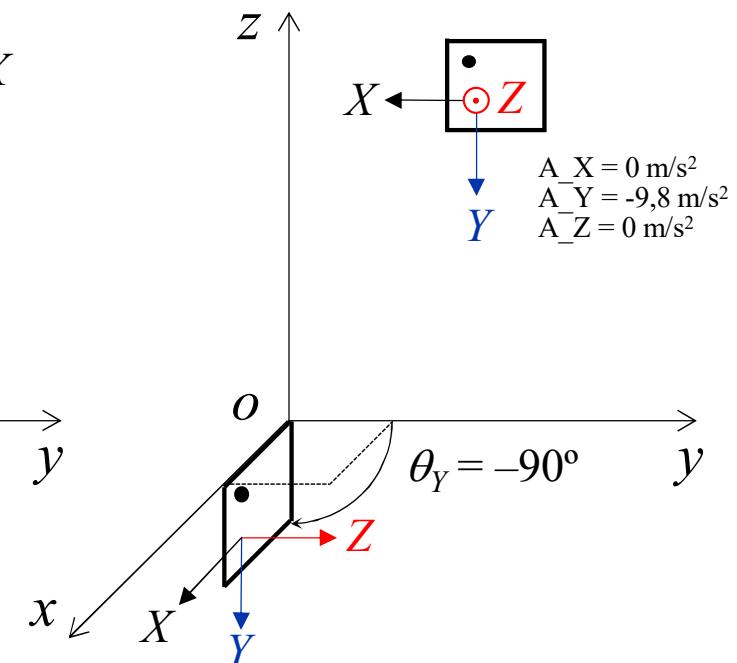
$$0 \text{ g em } Y \rightarrow \sin \theta_Y = 0$$



$$+1 \text{ g em } Y \rightarrow \sin \theta_Y = +1$$



$$-1 \text{ g em } Y \rightarrow \sin \theta_Y = -1$$





□ Utilização do acelerómetro para medição de inclinação

- Determinação do ângulo de inclinação

Nas posições intermédias que o acelerómetro toma, o valor da aceleração dada pelo acelerómetro varia em função da sua inclinação.

Por exemplo, para Y (em torno de x) :

$$\sin \theta_Y = 0 : A_Y = 0$$

$$\sin \theta_Y = +1 : A_Y = +9.8 \text{ m/s}^2$$

$$\sin \theta_Y = -1 : A_Y = -9.8 \text{ m/s}^2$$

O ângulo de inclinação do acelerómetro, em relação ao eixo Y , será dado por:

Pode inferir-se que:

$$\sin \theta_Y = \frac{A_Y}{9.8 \text{ m/s}^2}$$

$$\theta_Y = \arcsen \left(\frac{A_Y}{9.8 \text{ m/s}^2} \right)$$

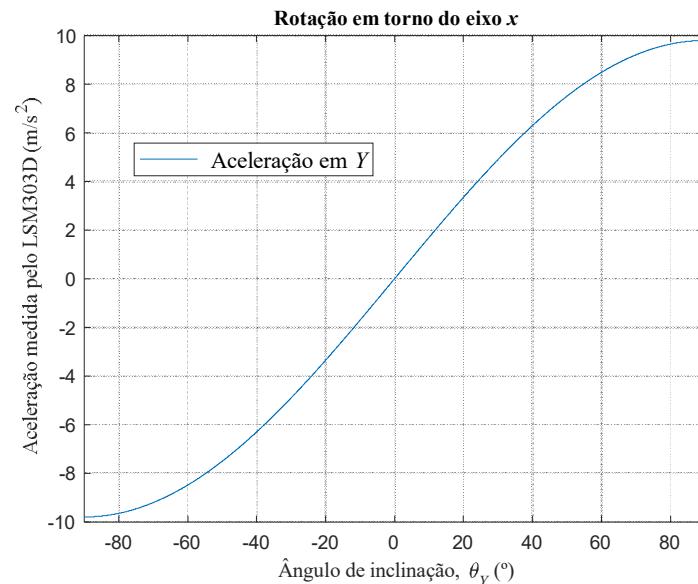


□ Utilização do acelerómetro para medição de inclinação

- Determinação do ângulo de inclinação

A abordagem anterior, apesar de funcionar para qualquer um dos três eixos, tem um problema: a sensibilidade é variável, sendo máxima na vizinhança de 0° e mínima (≈ 0), em $+90^\circ$ e -90° .

$$\text{sen } \theta_Y = \frac{A_Y}{9.8 \text{ m/s}^2} \Leftrightarrow A_Y = \text{sen } \theta_Y \times 9.8 \text{ m/s}^2$$



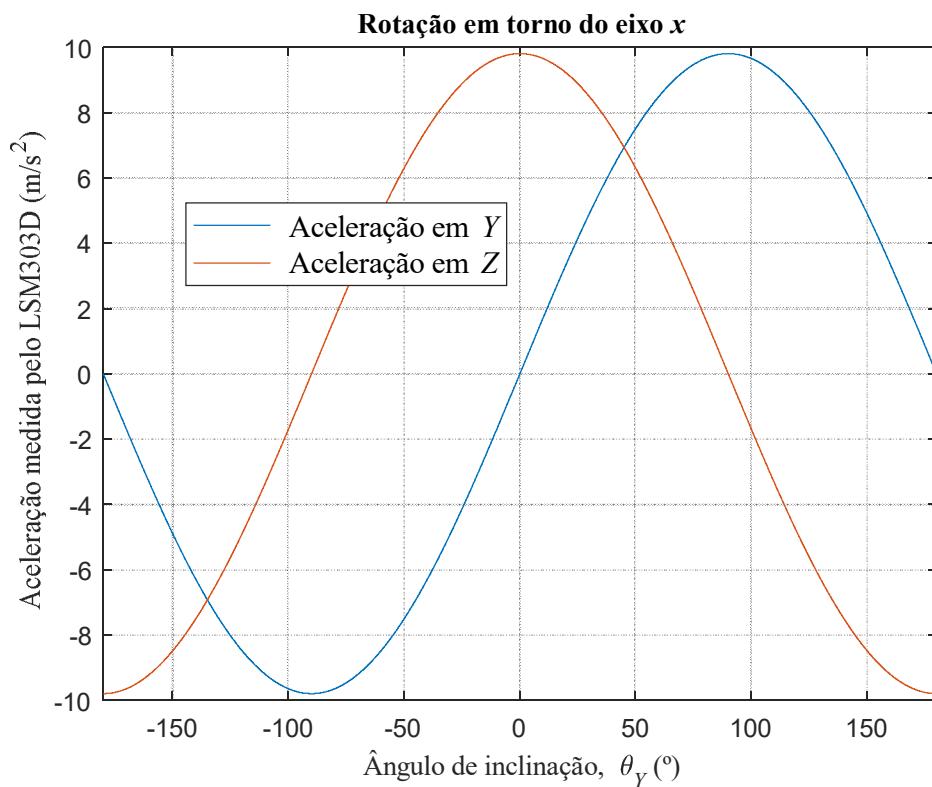


□ Utilização do acelerómetro para medição de inclinação

- Determinação do ângulo de inclinação

Uma forma de eliminar o problema da sensibilidade, é usar como auxílio um segundo eixo, ortogonal ao primeiro.

Desta forma, enquanto que a sensibilidade segundo um dos eixos está a diminuir, segundo o outro eixo, está a aumentar, dada a ortogonalidade entre os eixos.

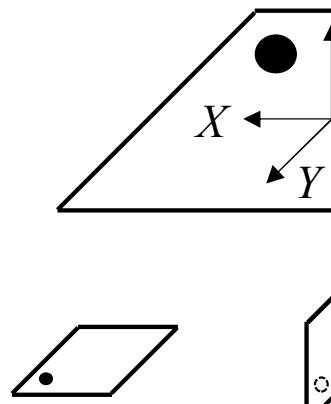




□ Utilização do acelerómetro para medição de inclinação

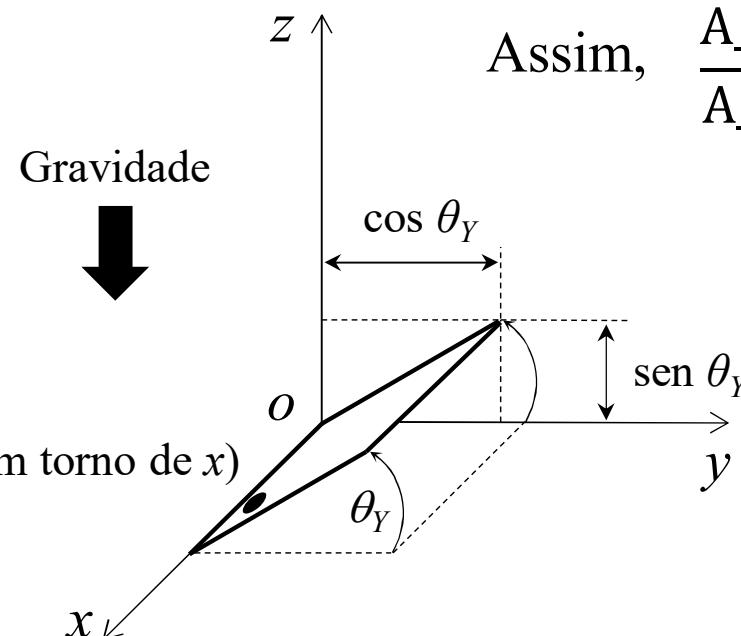
- Determinação do ângulo de inclinação

Por exemplo, considerando o sistema de eixos zoy , a gravidade segundo o eixo Y do acelerómetro é proporcional a $\sin \theta_Y$ e, segundo o eixo Z , é proporcional a $\cos \theta_Y$.



$$\begin{aligned}A_X &= 0 \text{ m/s}^2 \\ A_Y &= 0 \text{ m/s}^2 \\ A_Z &= 9,8 \text{ m/s}^2\end{aligned}$$

$$\begin{aligned}A_X &= 0 \text{ m/s}^2 \\ A_Y &= 9,8 \text{ m/s}^2 \\ A_Z &= 0 \text{ m/s}^2\end{aligned}$$



(rotação em torno de x)

$$\text{Assim, } \frac{A_Y}{A_Z} = \frac{\sin \theta_Y}{\cos \theta_Y} = \tan \theta_Y$$

Portanto,

$$\theta_Y = \arctan\left(\frac{A_Y}{A_Z}\right)$$

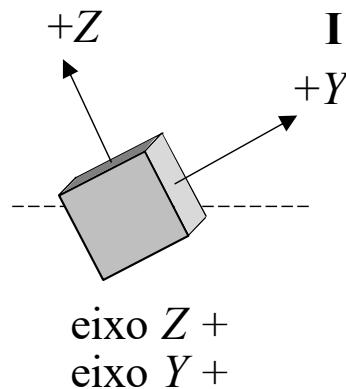


□ Utilização do acelerómetro para medição de inclinação

- Determinação do ângulo de inclinação

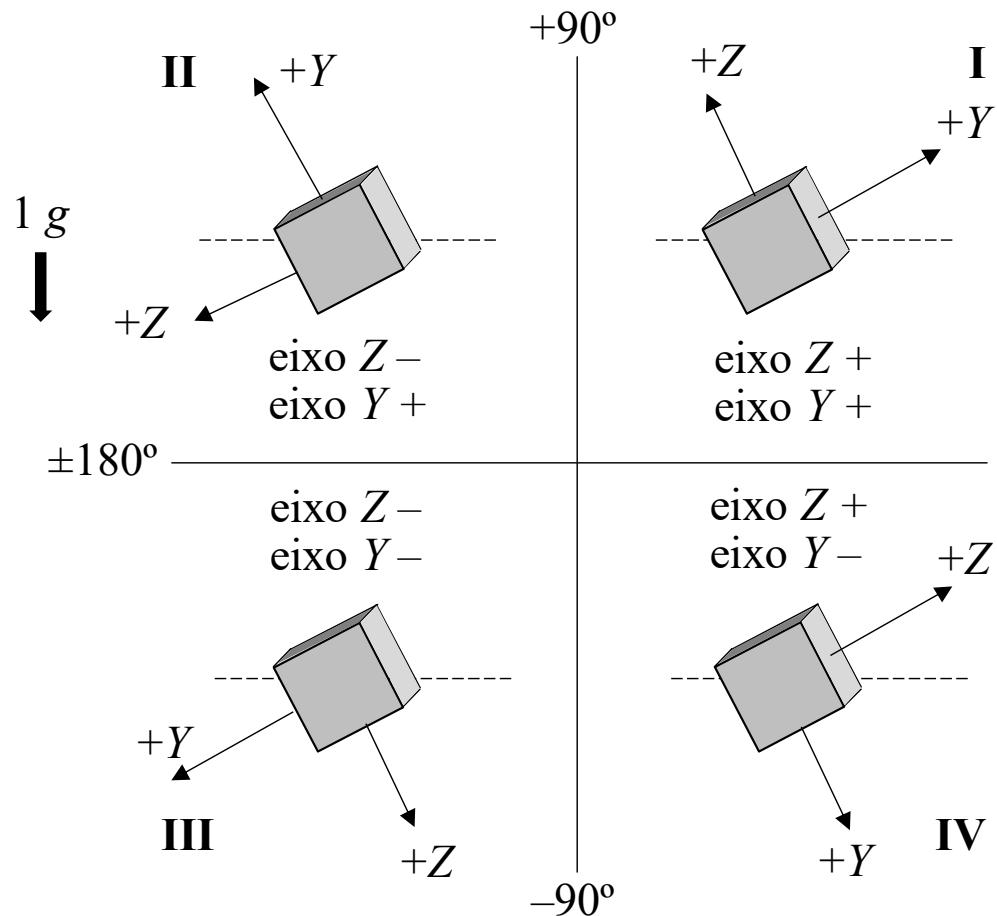
O método anterior só fornece valores no I e no IV quadrantes, portanto, ter-se-á uma medição entre $+90^\circ$ e -90° . Com dois eixos, também se sabe qual o quadrante em que se está, permitindo a medição de ângulos a $\pm 180^\circ$ no plano, dado o sinal algébrico da aceleração em cada eixo.

Eixos de gravidade do acelerómetro, relativamente ao I quadrante:



□ Utilização do acelerómetro para medição de inclinação

- Determinação do ângulo de inclinação



No II e III quadrantes, ao valor obtido pelo método já conhecido, deverá somar-se $+180^\circ$ e -180° , respetivamente, obtendo-se, no geral, ângulos que vão de 0 a $+180^\circ$ e de 0 a -180° . Introduzindo um terceiro eixo, seria possível determinar, por completo, a orientação do sensor no espaço (numa esfera: 3D).



□ Terminologia – magnetómetro

- Sensibilidade do sensor magnético

Descreve o ganho do sensor, *i.e.* a relação entre o valor indicado pela medição e o verdadeiro valor do campo magnético. Pode ser verificado, lendo os valores medidos, ao aplicar um campo magnético 1 gauss ($100 \mu\text{T}$).

O **fator de sensibilidade da indução magnética** varia de acordo com o fim de escala escolhido, sendo de **0,080** m gauss/LSB, **0,160** m gauss/LSB, **0,320** m gauss/LSB e **0,479** m gauss/LSB, para as escalas de **± 2** gauss, **± 4** gauss, **± 8** gauss e **± 12** gauss, respetivamente.



□ Terminologia – magnetómetro

- Nível de Zero-gauss

O *offset* relativamente ao nível de zero-gauss descreve o desvio do valor lido face à medição ideal quando não há campo magnético aplicado. Graças ao impulso de *set/reset* e à cadeia de medida, este *offset* é automaticamente cancelado.

O LSM303D vem calibrado de fábrica. Os valores de ajuste estão guardados numa memória não-volátil do dispositivo. Cada vez que este é ligado, os valores são colocados nos registos para serem usados durante a operação normal, levando o utilizador a não ter que realizar mais calibrações.



☐ Registos de controlo mais relevantes do LSM303D – magnetómetro

- CTRL5 – Definição da resolução e do ODR do sensor magnético (ver tabelas 46 e 47).

Table 45. CTRL5 register

TEMP_EN	M_RES1	M_RES0	M_ODR2	M_ODR1	M_ODR0	LIR2	LIR1
---------	--------	--------	--------	--------	--------	------	------

- CTRL6 – Definição do fim de escala do sensor magnético (ver tabelas 49 e 50).

Table 48. CTRL6 register

0 ⁽¹⁾	MFS1	MFS0	0 ⁽¹⁾				
------------------	------	------	------------------	------------------	------------------	------------------	------------------

1. These bits must be set to '0' for the correct working of the device.

- CTRL7 – Definições de filtragem do acelerómetro e de *low-power* do sensor magnético (ver tabelas 52, 53 e 54).

Table 51. CTRL7 register

AHPM1	AHPM0	AFDS	T_ONLY	0 ⁽¹⁾	MLP	MD1	MD0
-------	-------	------	--------	------------------	-----	-----	-----

1. This bit must be set to '0' for the correct working of the device.



☐ Registos de controlo mais relevantes do LSM303D – magnetómetro

- STATUS_M – Indicadores (*flags*) sobre se existe nova informação disponível (*data available* – DA) a partir do magnetómetro, ou se houve dados que foram medidos, mas que não foram entretanto lidos, tendo sido esmagados (*overrun* – OR). Estes indicadores são individualizados por eixo, havendo ainda para DA ou OR, um bit apenas que indica a ocorrência de cada uma das situações anteriores em algum dos eixos (ver tabela 18).

Se o bit T_ONLY do registo CTRL7 estiver ativo, também é sinalizado o OR ou DA da temperatura.

Table 17. STATUS_M register

ZYXMOR/ Tempor	ZMOR	YMOR	XMOR	ZYXMDA / Tempda	ZMDA	YMDA	XMDA
----------------	------	------	------	-----------------	------	------	------



☐ Registros de controlo mais relevantes do LSM303D – magnetómetro

- OUT_X_H_M e OUT_X_L_M – Informação a oito bits sobre a parte alta e a parte baixa da informação magnética digitalizada, sob o eixo do X.
- OUT_Y_H_M e OUT_Y_L_M – Informação a oito bits sobre a parte alta e a parte baixa da informação magnética digitalizada, sob o eixo do Y.
- OUT_Z_H_M e OUT_Z_L_M – Informação a oito bits sobre a parte alta e a parte baixa da informação magnética digitalizada, sob o eixo do Z.

A informação obtida deverá ser **convenientemente concatenada** e **multiplicada pelo fator de sensibilidade**, de acordo com o **fim de escala** escolhido.



☐ Registros de dados de temperatura do LSM303D

No LSM303D existe também um par de registos, a partir do qual se pode obter o valor da temperatura ambiente.

- TEMP_OUT_H e TEMP_OUT_L – Informação a oito bits, expressa em complemento para 2, sobre a temperatura do ambiente onde o sensor está colocado. Estes registos devem ser concatenados de maneira a formar um valor a 12 bits, alinhado à direita.

Pode ativar-se esta possibilidade fazendo *set* ao bit TEMP_EN do registo CTRL5.



Sequência de funcionamento para o magnetómetro

1. Inicialização do magnetómetro, nomeadamente, verificando a identificação do circuito integrado e escrevendo os valores adequados nos registos que tenham que funcionar com valores diferentes dos de *default*, por exemplo, definindo o fim de escala.

2. Leitura da informação a 16 bits (inteiros com sinal), relativa à densidade de fluxo magnético, ou indução magnética, sob um determinado eixo (X , Y ou Z), multiplicada pelo fator de sensibilidade inerente ao fim de escala em vigor.



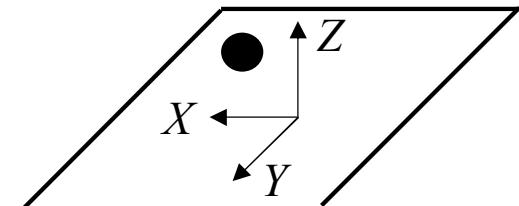
□ Funções de interação com o magnetómetro do LSM303D

Funções de acesso básico (as mesmas do acelerómetro):

- byte LSM303D_read_8bit_value(byte regAddress)
- void LSM303D_write_8bit_value(byte regAddress, byte value)

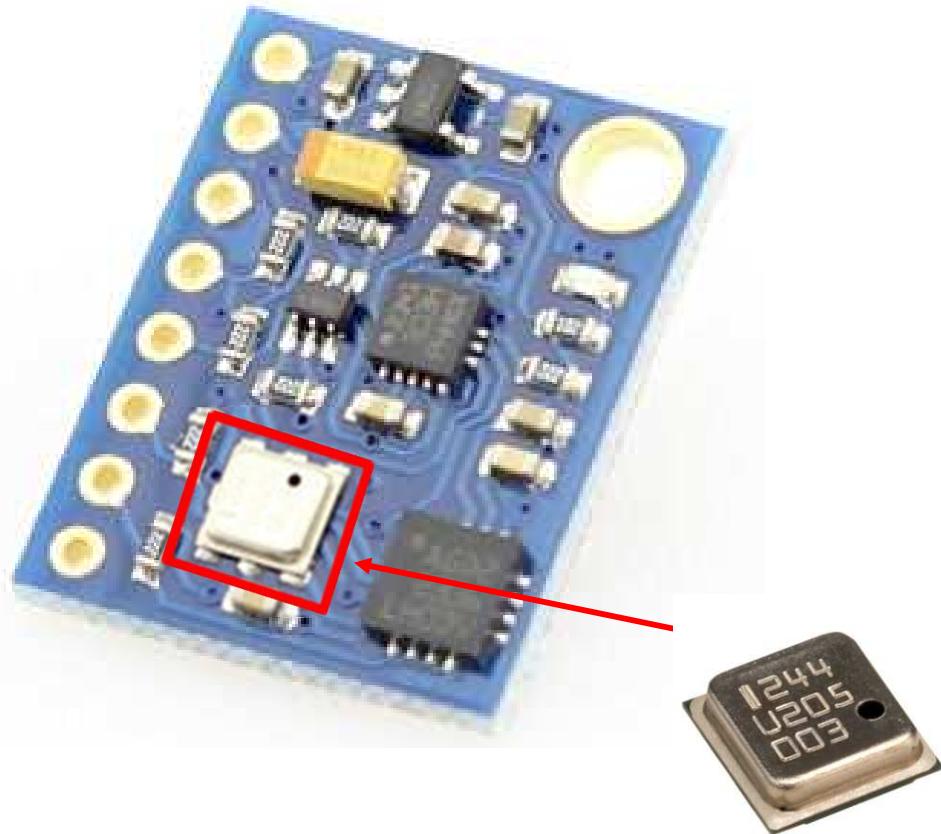
Funções de inicialização e leitura de informação do magnetómetro:

- bool LSM303D_M_Init(byte senseRange)
- float LSM303D_M_X_channel_Read()
- float LSM303D_M_Y_channel_Read()
- float LSM303D_M_Z_channel_Read()





□ Introdução



BMP180

BMP180 - Sensor de pressão
atmosférica (hPa) e temperatura

(°C) com interface I²C.

Utilizado em *smartphones*, dispositivos de navegação GPS e equipamento para *outdoor*.

Mediante cálculos, também é possível determinar a altitude (m), através do valor da pressão atmosférica, compensada pela temperatura ambiente.



☐ Datasheet do BMP180

A utilização do sensor deve ser feita recorrendo às informações que constam no manual fornecido pelo fabricante.

No manual, encontram-se todas as características do sensor e os procedimentos a adotar para obter as informações de temperatura, de pressão atmosférica e de altitude.

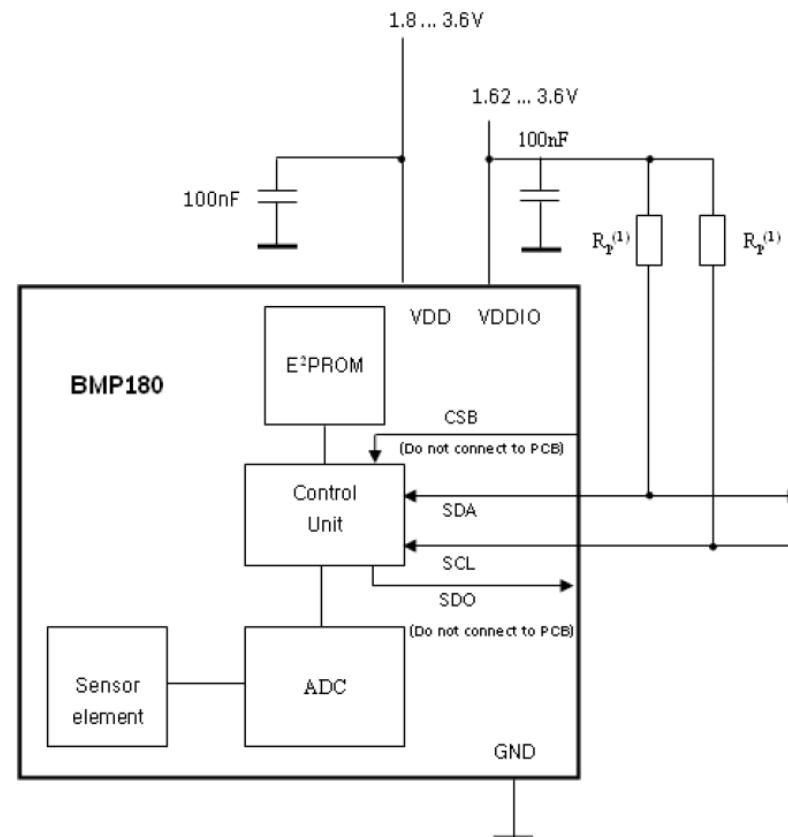




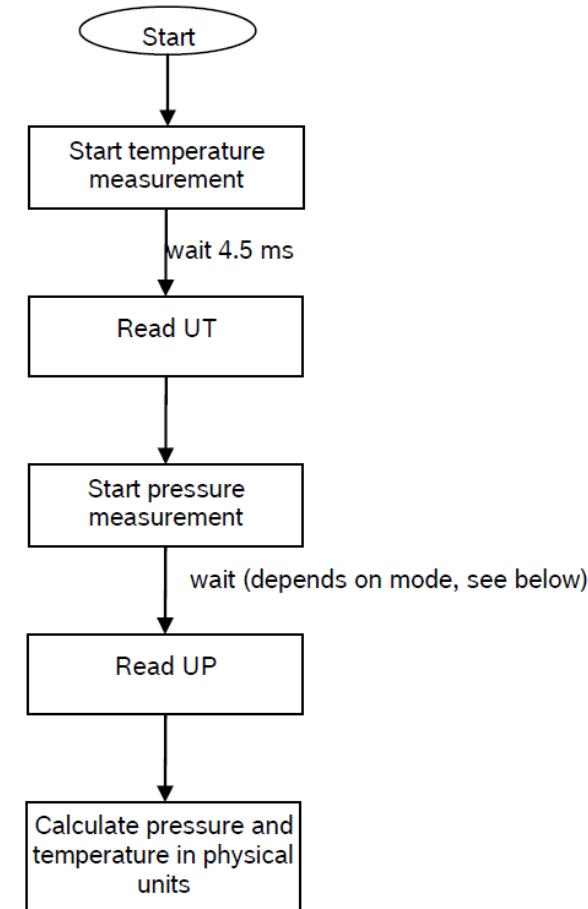
□ Introdução

- A interligação ao Arduino torna-se extremamente simples, graças à interface I²C, em que o endereço típico do sensor é 0x77
- O BMP180 consiste num sensor piezorresistivo, um conversor de analógico para digital (ADC), uma unidade de controlo com E²PROM e uma interface I²C
- A pressão e a temperatura não-compensadas têm que ser compensadas com dados de calibração existentes na E²PROM do BMP180
- A E²PROM guarda 11 parâmetros para compensar a temperatura e a pressão lidas “em bruto” (não-compensadas)

☐ Medições de temperatura e pressão



Esquema de ligações do módulo



Fluxo de procedimentos nas medições

☐ Modos *hardware* de precisão de amostragem de pressão

Mode	Parameter <i>oversampling_setting</i>	Internal number of samples	Conversion time pressure max. [ms]	Avg. current @ 1 sample/s typ. [μ A]	RMS noise typ. [hPa]	RMS noise typ. [m]
ultra low power	0	1	4.5	3	0.06	0.5
standard	1	2	7.5	5	0.05	0.4
high resolution	2	4	13.5	7	0.04	0.3
ultra high resolution	3	8	25.5	12	0.03	0.25

Modos de
oversampling

Valores possíveis do parâmetro
oversampling_setting

Tempo de conversão inerente a cada
modo do *oversampling* (*delay*)



☐ Registros internos do BMP180

Register Name	Register Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset
out xlsb	F8h			adc_out_xlsb<7:3			0	0	0	00h
out lsb	F7h				adc_out_msb<7:0>					00h
out msb	F6h					adc_out_lsb<7:0				80h
ctrl_meas	F4h	oss<1:0>		sco			measurement			00h
soft	E0h				reset					00h
id	D0h				id<7:0>					55h
calib21 downto calib0	BFh downto AAh				calib21<7:0> downto calib0<7:0>					n/a

Registers: Control Calibration Dat
Type: registers registers registers Fixed

Registros com os coeficientes de calibração (leitura): 0xAA a 0xBF

Registo de controlo das medições (escrita): 0xF4

Registo de resposta de medições não-compensadas (leitura): 0xF6 a 0xF8

Leitura de UT: 2 bytes
(Uncompensated Temperature)

Leitura de UP: 3 bytes
(Uncompensated Pressure)



☐ Valores do registo de controlo

De acordo com a medição pretendida, existem vários valores que devem ser colocados no registo de controlo (no endereço 0xF4), bem como diferentes intervalos de tempo que se tem que aguardar para que a conversão se realize.

Measurement	Control register value (register address 0xF4)	Max. conversion time [ms]
Temperature	0x2E	4.5
Pressure (oss = 0)	0x34	4.5
Pressure (oss = 1)	0x74	7.5
Pressure (oss = 2)	0xB4	13.5
Pressure (oss = 3)	0xF4	25.5



□ Coeficientes de calibração

A E²PROM contém 11 *words* de 16 bit (short) com os 11 coeficientes de calibração. Antes dos cálculos de temperatura e pressão, estes dados são lidos.

Parameter	BMP180 reg adr	
	MSB	LSB
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xB0	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

O endereço base de cada coeficiente é o que corresponde ao MSB, tendo sempre que ser pedidos 2 bytes ao sensor por cada coeficiente, de acordo com o procedimento para o I²C, usando o método `Wire.requestFrom()`.

O valor obtido no final resulta da concatenação dos 2 bytes, devidamente alinhados, para formar uma *word* de 16 bits.



☐ Cálculo de pressão e de temperatura

Possíveis modos de *oversampling*:

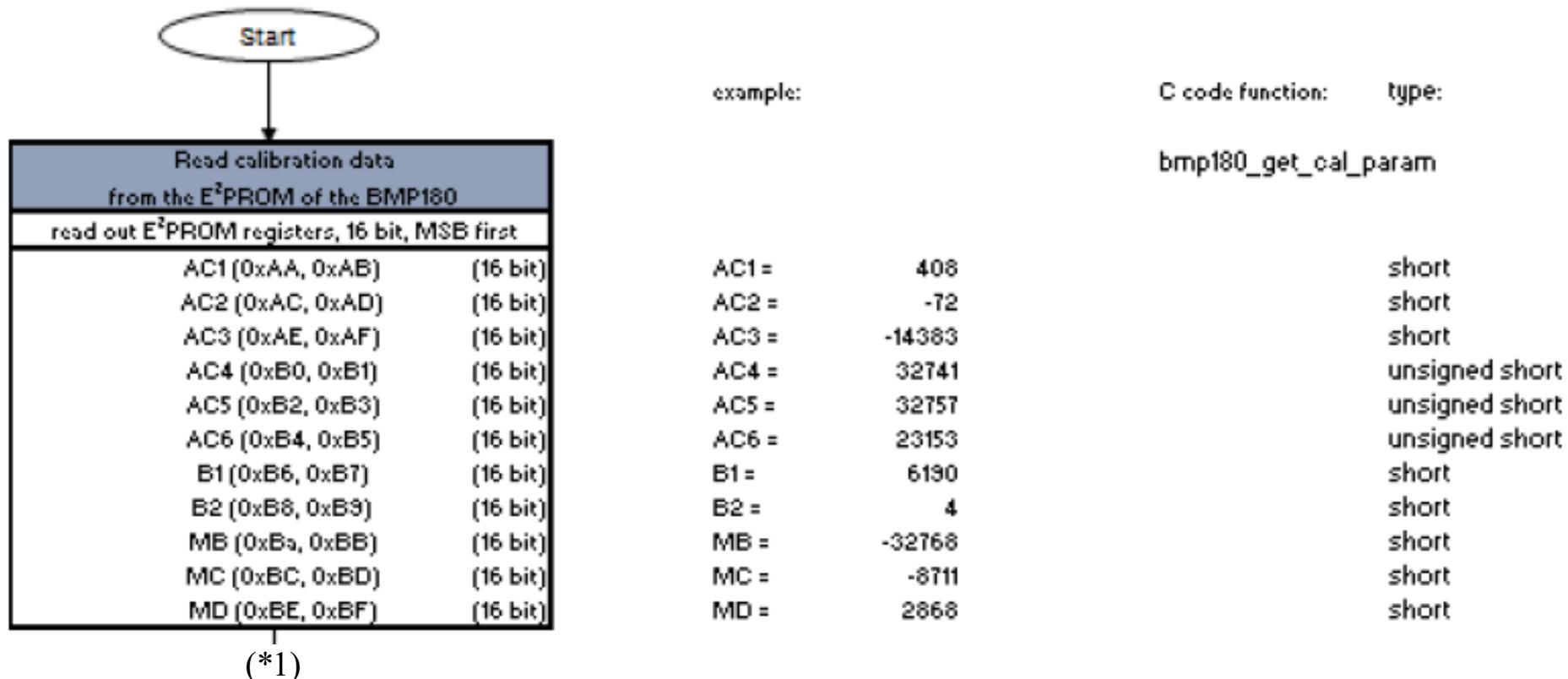
- ultra low power (0)
- standard (1)
- high resolution (2)
- ultra high resolution (3)

Um destes modos tem que ser escolhido, atribuindo o valor adequado à variável que contém o *oversampling setting* (oss).

O cálculo da pressão real é feito em unidades de 1Pa (= 0.01hPa = 0.01mbar) e o da temperatura real, em unidades de 0.1°C.

□ Algoritmo para medição de pressão e de temperatura reais

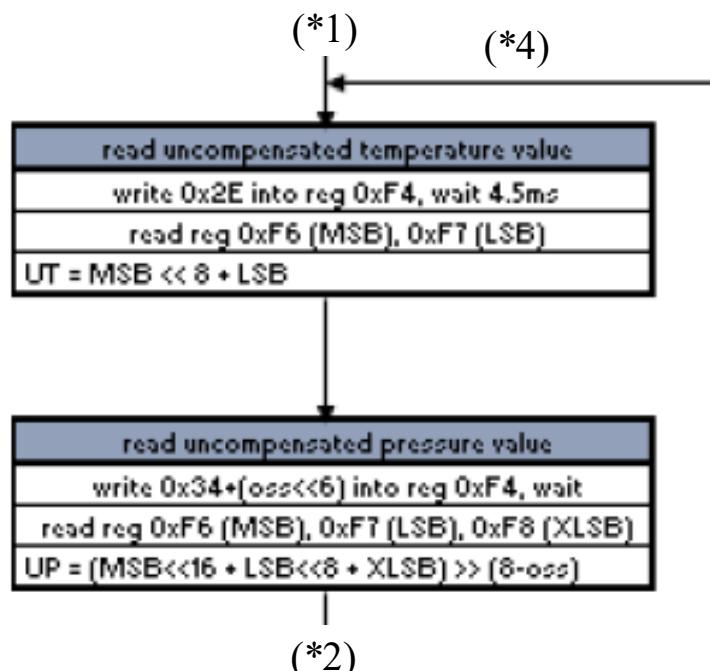
1.º passo: Leitura dos coeficientes de calibração do sensor. Esta leitura é realizada apenas uma vez, servindo para todas as medições subsequentes.





☐ Algoritmo para medição de pressão e de temperatura reais

2.º passo: Leitura dos valores da temperatura e da pressão não-compensadas, escrevendo no endereço 0xF4, esperando um tempo relacionado com a medição a realizar, e lendo dos endereços 0xF6 a 0xF8 a informação para cada caso.

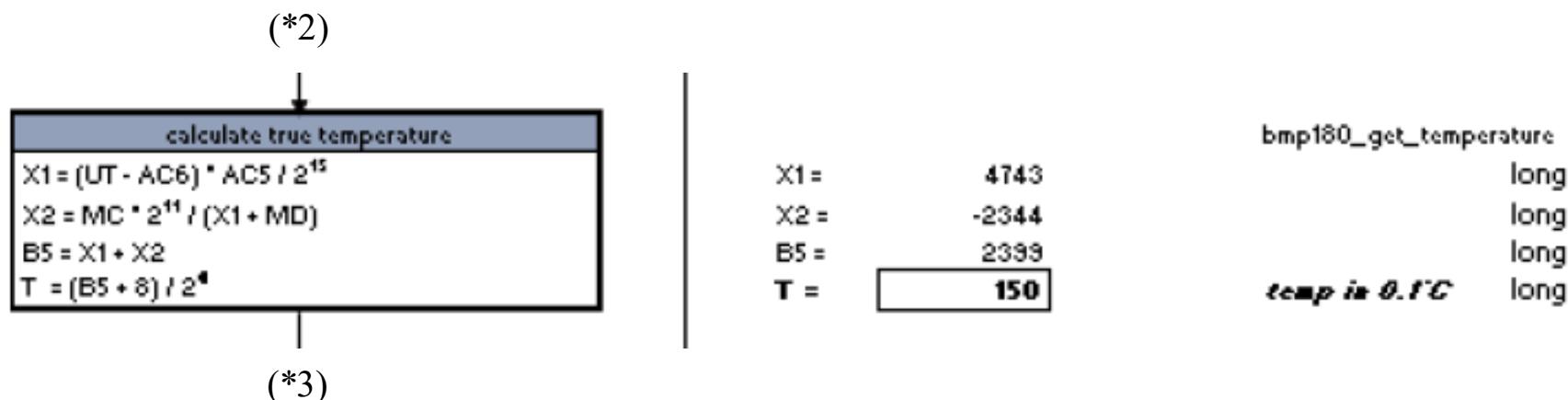


bmp180_get_ut	long
UT = 27898	
oss = 0 = oversampling_setting (ultra low power mode)	short (0 .. 3)
bmp180_get_up	
UP = 23843	long



☐ Algoritmo para medição de pressão e de temperatura reais

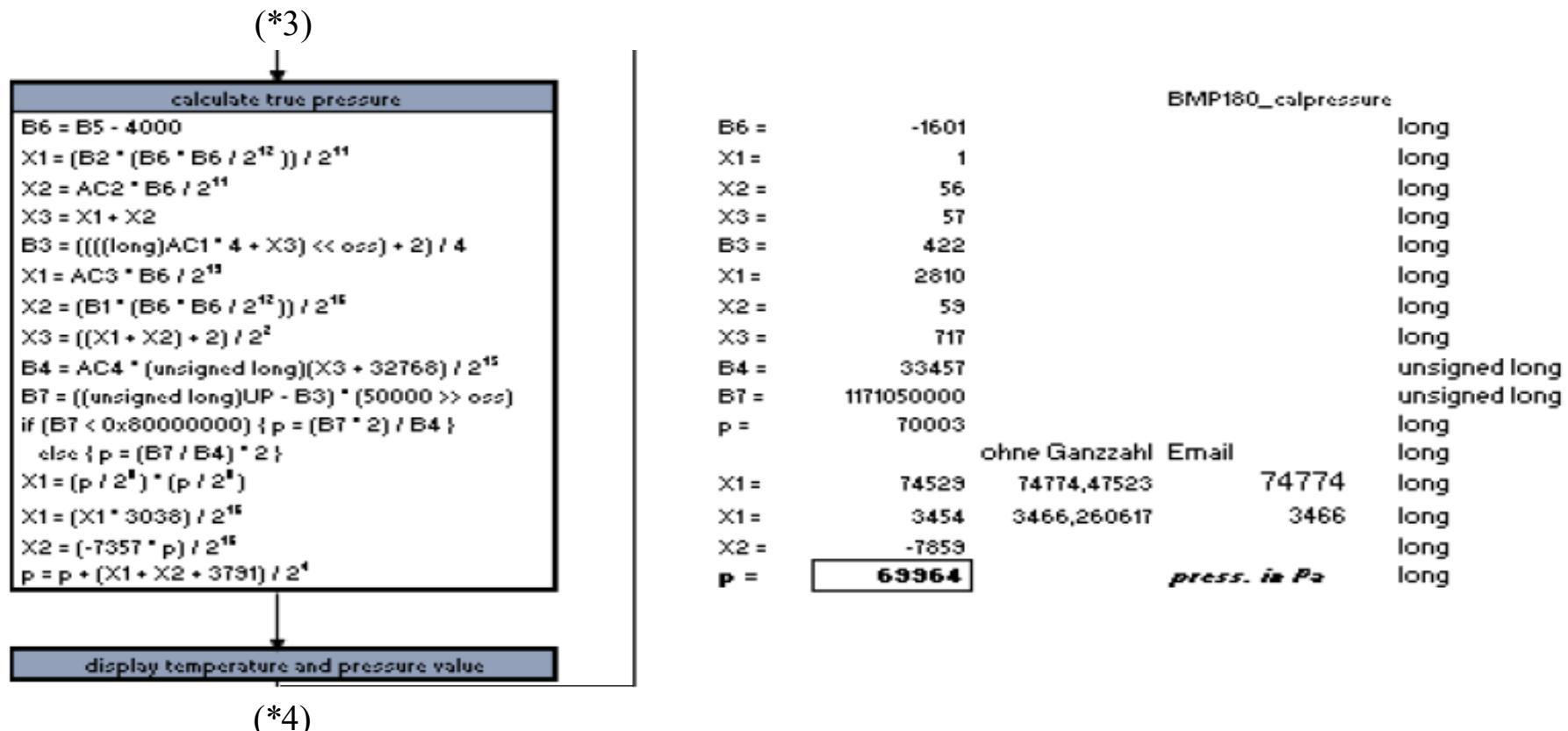
3.º passo: Cálculo do valor da temperatura real. Tendo como base o valor da temperatura não-compensada, realizando os devidos cálculos, obtém-se o valor da temperatura real, estando esta em unidades de 0,1°C. Os parâmetros de calibração envolvidos são AC5, AC6, MC e MD. Este passo pode e deve juntar-se com o passo seguinte numa única função no código.





□ Algoritmo para medição de pressão e de temperatura reais

4.^º passo: Cálculo do valor de pressão real. Com a pressão não-compensada, obtém-se o valor da pressão real, recorrendo aos coeficientes de calibração.



□ Cálculo da altitude absoluta

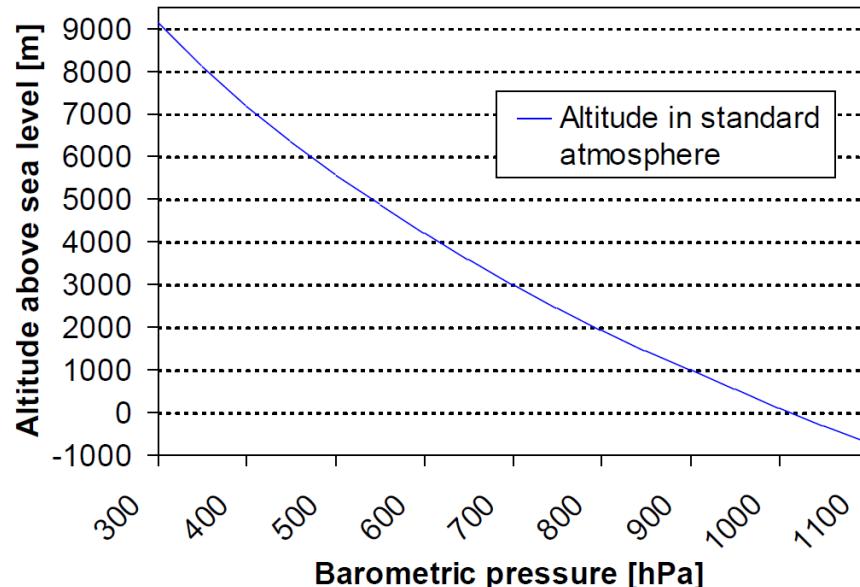
Com o valor da pressão real, juntamente com o valor da pressão atmosférica ao nível do mar, é possível estimar o valor da altitude (medida em metros), utilizando a seguinte expressão:

$$h = 44300 \times \left(1 - \left(\frac{p}{p_0} \right)^{\frac{1}{5,255}} \right)$$

h – altitude (m)

p – pressão atmosférica medida (hPa)

p_0 – pressão atmosférica ao nível do mar, por exemplo, 1013,25 hPa





□ Funções de interação com o BMP180

Funções de acesso básico:

- short BMP180_read_16bit_value(byte regAddress)
- long BMP180_read_24bit_value(byte regAddress)
- void BMP180_write_8bit_value(byte regAddress, byte value)

Funções de inicialização e leitura de informação:

- void BMP180_readCalibrationData()
- short BMP180_readUncompensatedTemperatureValue()
- long BMP180_readUncompensatedPressureValue()
- void BMP180_calculateTrueTemperatureAndPressure()
- float BMP180_calculateAltitude()



ISEL – Instituto Superior de Engenharia de Lisboa

*ADEETC – Área Departamental de Engenharia de Electrónica e
Telecomunicações e de Computadores*

LEIM – Licenciatura em Engenharia Informática e Multimédia

Comunicação série

Comunicação série



□ Comunicação série no Arduino

A placa do Arduino dispõe de um canal de comunicação série com o computador, através do qual, por exemplo, são carregados os programas.

Níveis lógicos: “1”: 5 V “0”: 0 V

É necessário haver um *chip* para converter os sinais do porto de *hardware* série do Arduino para USB (*Universal Serial Bus*), por exemplo, o FTDI232.

O protocolo RS-232 é já utilizado há muito tempo, mas ainda é usado correntemente. Os seus níveis de tensão são diferentes dos do Arduino, por isso, é necessário ter um circuito conversor de tensão.



□ Comunicação série no Arduino

- Para que exista uma comunicação efetiva entre duas máquinas, é necessário que ambas obedeçam a uma forma organizada de estabelecimento de mensagens, tanto para envio, como para receção.
- A organização formal da informação numa mensagem e a gama de respostas apropriadas a determinados pedidos, denomina-se por protocolo de comunicação.
- O protocolo série *start-stop* trata do envio dos vários caracteres que compõem o código ASCII do carácter a enviar.

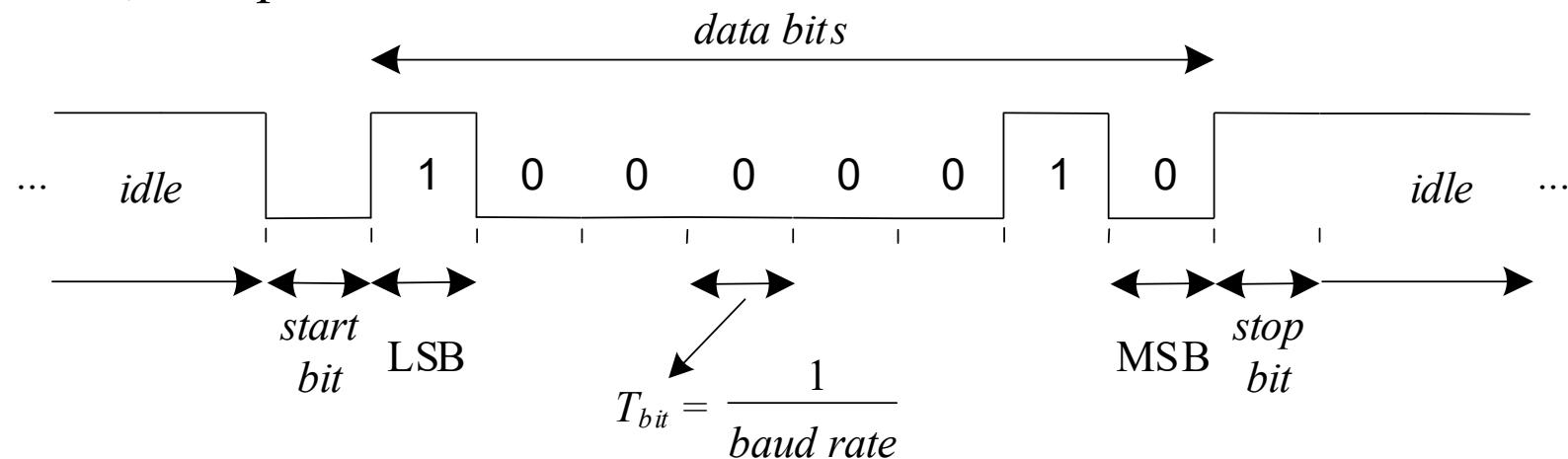


☐ Tabela ASCII

Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char
0	0000 0000	00	[NUL]	32	0010 0000	20	space	64	0100 0000	40	€	96	0110 0000	60	`
1	0000 0001	01	[SOH]	33	0010 0001	21	!	65	0100 0001	41	À	97	0110 0001	61	à
2	0000 0010	02	[STX]	34	0010 0010	22	"	66	0100 0010	42	ß	98	0110 0010	62	ß
3	0000 0011	03	[ETX]	35	0010 0011	23	#	67	0100 0011	43	©	99	0110 0011	63	©
4	0000 0100	04	[EOT]	36	0010 0100	24	\$	68	0100 0100	44	Ð	100	0110 0100	64	ð
5	0000 0101	05	[ENQ]	37	0010 0101	25	%	69	0100 0101	45	£	101	0110 0101	65	£
6	0000 0110	06	[ACK]	38	0010 0110	26	&	70	0100 0110	46	ƒ	102	0110 0110	66	ƒ
7	0000 0111	07	[BEL]	39	0010 0111	27	.	71	0100 0111	47	„	103	0110 0111	67	„
8	0000 1000	08	[BS]	40	0010 1000	28	(72	0100 1000	48	H	104	0110 1000	68	H
9	0000 1001	09	[TAB]	41	0010 1001	29)	73	0100 1001	49	I	105	0110 1001	69	I
10	0000 1010	0A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0B	[VT]	43	0010 1011	2B	+	75	0100 1011	4B	K	107	0110 1011	6B	k
12	0000 1100	0C	[FF]	44	0010 1100	2C	,	76	0100 1100	4C	L	108	0110 1100	6C	l
13	0000 1101	0D	[CR]	45	0010 1101	2D	-	77	0100 1101	4D	M	109	0110 1101	6D	m
14	0000 1110	0E	[SO]	46	0010 1110	2E	.	78	0100 1110	4E	N	110	0110 1110	6E	n
15	0000 1111	0F	[SI]	47	0010 1111	2F	/	79	0100 1111	4F	O	111	0110 1111	6F	o
16	0001 0000	10	[DLE]	48	0011 0000	30	0	80	0101 0000	50	P	112	0111 0000	70	p
17	0001 0001	11	[DC1]	49	0011 0001	31	1	81	0101 0001	51	Q	113	0111 0001	71	q
18	0001 0010	12	[DC2]	50	0011 0010	32	2	82	0101 0010	52	R	114	0111 0010	72	r
19	0001 0011	13	[DC3]	51	0011 0011	33	3	83	0101 0011	53	S	115	0111 0011	73	s
20	0001 0100	14	[DC4]	52	0011 0100	34	4	84	0101 0100	54	T	116	0111 0100	74	t
21	0001 0101	15	[NAK]	53	0011 0101	35	5	85	0101 0101	55	U	117	0111 0101	75	u
22	0001 0110	16	[SYN]	54	0011 0110	36	6	86	0101 0110	56	V	118	0111 0110	76	v
23	0001 0111	17	[ETB]	55	0011 0111	37	7	87	0101 0111	57	W	119	0111 0111	77	w
24	0001 1000	18	[CAN]	56	0011 1000	38	8	88	0101 1000	58	X	120	0111 1000	78	x
25	0001 1001	19	[EM]	57	0011 1001	39	9	89	0101 1001	59	Y	121	0111 1001	79	y
26	0001 1010	1A	[SUB]	58	0011 1010	3A	:	90	0101 1010	5A	Z	122	0111 1010	7A	z
27	0001 1011	1B	[ESC]	59	0011 1011	3B	:	91	0101 1011	5B	[123	0111 1011	7B	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	\	124	0111 1100	7C	
29	0001 1101	1D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D]	125	0111 1101	7D	}
30	0001 1110	1E	[RS]	62	0011 1110	3E	>	94	0101 1110	5E	^	126	0111 1110	7E	~
31	0001 1111	1F	[US]	63	0011 1111	3F	?	95	0101 1111	5F	_	127	0111 1111	7F	[DEL]

□ Comunicação série no Arduino

- O envio dos bits dos caracteres inicia-se pelo LSB.
- O tempo de duração de cada bit na linha de transmissão é o inverso do *baud rate* (ritmo de envio de dados) escolhido para a comunicação.
- Exemplo: envio de um carácter ‘A’ ($b01000001 = 0x41$), 8 *data bits*, sem paridade, 1 *stop bit*





☐ Tabela ASCII estendida

ASCII control characters			ASCII printable characters								Extended ASCII characters							
00	NULL	(Null character)	32	space	64	@	96	.	128	ç	160	á	192	l	224	ó		
01	SOH	(Start of Header)	33	!	65	A	97	a	129	ü	161	í	193	ł	225	b		
02	STX	(Start of Text)	34	"	66	B	98	b	130	é	162	ö	194	ł	226	ó		
03	ETX	(End of Text)	35	#	67	C	99	c	131	â	163	ú	195	ł	227	ó		
04	EOT	(End of Trans.)	36	\$	68	D	100	d	132	ã	164	ñ	196	—	228	ø		
05	ENQ	(Enquiry)	37	%	69	E	101	e	133	à	165	Ñ	197	+	229	ò		
06	ACK	(Acknowledgement)	38	&	70	F	102	f	134	â	166	»	198	á	230	µ		
07	BEL	(Bell)	39	·	71	G	103	g	135	ç	167	°	199	À	231	þ		
08	BS	(Backspace)	40	(72	H	104	h	136	è	168	¿	200	Ł	232	þ		
09	HT	(Horizontal Tab)	41)	73	I	105	i	137	ë	169	®	201	Ł	233	ú		
10	LF	(Line feed)	42	*	74	J	106	j	138	è	170	¬	202	Ł	234	ó		
11	VT	(Vertical Tab)	43	+	75	K	107	k	139	í	171	½	203	Ł	235	ú		
12	FF	(Form feed)	44	,	76	L	108	l	140	í	172	¼	204	Ł	236	ý		
13	CR	(Carriage return)	45	-	77	M	109	m	141	í	173	i	205	=	237	Ý		
14	SO	(Shift Out)	46	.	78	N	110	n	142	Ã	174	«	206	+	238	-		
15	SI	(Shift In)	47	/	79	O	111	o	143	Ã	175	»	207	■	239	.		
16	DLE	(Data link escape)	48	0	80	P	112	p	144	É	176	—	208	ð	240	≡		
17	DC1	(Device control 1)	49	1	81	Q	113	q	145	æ	177	—	209	Đ	241	±		
18	DC2	(Device control 2)	50	2	82	R	114	r	146	Æ	178	—	210	È	242	¾		
19	DC3	(Device control 3)	51	3	83	S	115	s	147	ô	179	—	211	È	243	½		
20	DC4	(Device control 4)	52	4	84	T	116	t	148	ö	180	—	212	È	244	1		
21	NAK	(Negative acknowl.)	53	5	85	U	117	u	149	ò	181	À	213	ı	245	§		
22	SYN	(Synchronous idle)	54	6	86	V	118	v	150	û	182	Ã	214	ı	246	÷		
23	ETB	(End of trans. block)	55	7	87	W	119	w	151	ú	183	Ã	215	ı	247	.		
24	CAN	(Cancel)	56	8	88	X	120	x	152	ÿ	184	©	216	ı	248	°		
25	EM	(End of medium)	57	9	89	Y	121	y	153	ö	185	—	217	ı	249	..		
26	SUB	(Substitute)	58	:	90	Z	122	z	154	Ù	186	—	218	—	250	.		
27	ESC	(Escape)	59	:	91	[123	{	155	ø	187]	219	—	251	..		
28	FS	(File separator)	60	<	92	\	124		156	£	188]	220	—	252	..		
29	GS	(Group separator)	61	=	93]	125	}	157	Ø	189	¢	221	—	253	..		
30	RS	(Record separator)	62	>	94	^	126	~	158	×	190	¥	222	—	254	■		
31	US	(Unit separator)	63	?	95	—			159	f	191	ł	223	—	255	nbsp		
127	DEL	(Delete)							160		192		224					



□ Utilização do Python para comunicação série

- É possível ter uma aplicação realizada pelo utilizador, a correr no computador, servindo essa aplicação para comunicar com o Arduino. A intenção é ter-se um canal de comunicação entre o computador e o Arduino, sem recorrer ao *Serial Monitor*.
- Para tal, deve instalar-se a biblioteca “Serial” do Python, de modo a poder realizar aplicações que façam uso da comunicação série.



□ Utilização do Python para comunicação série

De modo a que seja possível o estabelecimento de um canal de comunicação série entre o Arduino e uma aplicação realizada pelo utilizador no PC, não pode ter-se o *Serial Monitor* do Arduino em execução. A acontecer, não seria possível estabelecer a ligação da aplicação no PC ao Arduino, pois o canal já estaria ocupado pela comunicação entre o *Serial Monitor* e o Arduino.

→ Deve primeiro carregar-se o programa no Arduino e só depois executar a aplicação do lado do PC, a qual deve estabelecer, por seu turno, a ligação ao Arduino.



□ Funções básicas em Python para comunicação série

```
# Programa para comunicar com o Arduino (em Python 2.7)

import serial

com = 'COM5'      # Por exemplo, mas tem de coincidir com o do Arduino
baudrate = 9600 # Por exemplo, mas tem de coincidir com o do Arduino

# Função de inicialização
def comInit(com, baudrate):
    try:
        Serie = serial.Serial(com, baudrate)
        print 'Sucesso na ligacao ao Arduino.'
        print 'Ligado ao ' + Serie.portstr
        return Serie
    except Exception as e:
        print 'Insucesso na ligacao ao Arduino.'
        print e
        return None
```



□ Funções básicas em Python para comunicação série

```
def caracterReceive(Serie):
    try:
        return Serie.read()
    except Exception as e:
        print 'Erro na comunicacao (caracterReceive).'
        print e
        Serie.close()

def caracterSend(Serie, info):
    try:
        Serie.write(info.encode('utf-8'))
    except Exception as e:
        print 'Erro de comunicacao (caracterSend).'
        print e
        Serie.close()

def stringReceive(Serie):
    try:
        return Serie.readline().strip()
    except Exception as e:
        print 'Erro na comunicacao (stringReceive).'
        print e
        Serie.close()
```



□ Funções básicas em Python para comunicação série

```
# Programa principal
s = comInit(com, baudrate)
print s
while (s != None):
    c = raw_input('Introduza caracter: ')
    caracterSend(s, c) # Enviar um caracter para o Arduino (Arduino é RX)
    print '[TX] PC -> Arduino [RX]: ' + c
    print stringReceive(s) # Receber uma string de resposta (Arduino é TX)
```

Do lado do Arduino, ter-se-á que ter um programa que utiliza a classe `Serial`, tanto para envio como para receção de informação através da porta série. O programa no Arduino recebe o carácter enviado pelo computador e envia uma *string* de volta, com o código do carácter seguinte ao recebido do computador. Em acréscimo, o estado do LED da *board* é comutado cada vez que o Arduino receber o carácter ‘L’.



□ Funções básicas em Python para comunicação série

Programa no Arduino, que recebe caracteres do PC, envia uma *string* para o computador e comuta o estado do LED da *board*, ao receber o carácter 'L' .

```
boolean estado;

void setup() {
    Serial.begin(9600);
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
    if (Serial.available()) {
        char c = Serial.read();
        Serial.print("[TX] Arduino -> PC [RX]: ");
        Serial.println((char)(c + 1));
        if (c == 'L')
            digitalWrite(LED_BUILTIN, estado = !estado);
    }
}
```



□ Bibliografia

- Carvalho, C. (2007). Apontamentos manuscritos sobre Sistemas Digitais
- Pais, J. (2017). Folhas de Computação Física, ISEL
- Pimenta Rodrigues, V. & Seia de Araújo, M. (2001). *Projecto de Sistemas Digitais*, Editorial Presença
- Wakerly, John F. (2001). *Digital Design – Principles and Practices (3rd edition)*, Prentice Hall
- Website www.arduino.cc
- Ilett, J., (1998). How to use Intelligent L.C.D.s, Wimborne Publishing Ltd.
- Cytron Technologies Sdn. Bhd. (2013). Product User's manual – HC-SR04 Ultrasonic Sensor, Cytron Technologies Incorporated
- NXP Semiconductors. (2012). I²C-bus specification and user manual (Rev. 5)
- ST Microelectronics. (2013). L3GD20 MEMS motion sensor: three-axis digital output gyroscope (Doc ID 022116 Rev 2)
- ST Microelectronics. (2012). LSM303D Ultra compact high performance e-Compass - 3D accelerometer and 3D magnetometer module (Doc ID 023312 Rev 1)
- Bosch Sensortec GmbH. (2015). BMP180 data sheet (Rev. 2.8)
- Wuxi I-CORE Electronics Co., Ltd. (2013). AiP31068 40 SEG / 16 COM Driver & Controller for Dot Matrix LCD/2 or 3 Line Serial Data Interface (Ver. 2013-03-A3)